**VŠB - Technická univerzita Ostrava**

**Fakulta elektrotechniky a informatiky**

**Katedra informatiky**

# Učení v Multiagentních systémech za pomocí symbolické reprezentace znalostí

# Learning in Multi-agent Systems and Symbolic Representation of Knowledge

**2019** **Adam Albert**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

# Zadání diplomové práce

Student: **Bc. Adam Albert**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Učení v Multiagentních systémech za pomocí symbolické reprezentace znalostí
Learning in Multi-agent Systems and Symbolic Representation of Knowledge

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je shrnout teorii strojového učení v multiagentních systémech za pomocí symbolické reprezentace poznatků. Student v rámci případové studie naprogramuje agenta, který bude na základě znalostí specifikovaných v jazyce TIL-Script rozšiřovat svou znalostní bázi za účelem získávání nových znalostí.

Cíle práce:
1. Shrnutí teorie učení v multiagentních systémech na základě symbolické reprezentace znalostí.
2. Práce bude obsahovat shrnutí teorie TIL a syntax jazyka TIL-Script.
3. Součástí práce bude taktéž analýza, návrh a implementace agenta schopného rozšiřovat svou znalostní bázi za účelem rozšíření svých schopností.

Seznam doporučené odborné literatury:

[1] Duží M., Materna P. (2012): TIL jako procedurální logika (průvodce zvídavého čtenáře Transparentní intensionální logikou). Aleph Bratislava 2012, ISBN 978-80-89491-08-7
[2] Duží M., Jespersen B. and Materna P. (2010): Procedural Semantics for Hyperintensional Logic. Foundations and Applications of Trasnsparent Intensional Logic. First edition. Berlin: Springer, series Logic, Epistemology, and the Unity of Science, vol. 17, ISBN 978-90-481-8811-6.
[3] Poole, D. L., Mackworth ,A. K.: Artificial Intelligence - foundations of computational agents, Cambridge 2017, ISBN: 978-1-107-19539-4

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Mgr. Marek Menšík, Ph.D.**

Datum zadání: 01.09.2018

Datum odevzdání: 30.04.2019

doc. Ing. Jan Platoš, Ph.D.
*vedoucí katedry*

prof. Ing. Pavel Brandštetter, CSc.
*děkan fakulty*

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30. dubna 2019

## Abstrakt

Tato práce se zaobírá strojovým učením se symbolickou reprezentací znalostí využívající Transparentní intenzionální logiku a jeho aplikací na prvku multiagentního systému. Symbolické metody strojového učení slouží jako nástroj pro zpracování zkušeností na formálně zapsané znalosti, které se využívají pro zvyšování inteligence softwarových aplikací, které tyto metody implementují. Práce shrnuje základy strojového učení, symbolické modely strojového učení, obsahuje teorii Transparentní intenzionální logiky a syntax jazyka pro její softwarově zpracovatelný zápis TIL-Script. Na závěr je uvedena analýza a návrh agenta implementující symbolickou metodu učení pomocí transparentní intenzionální logiky.



Klíčová slova: strojové učení, Transparentní intenzionální logika, TIL-Script, symbolické metody, multiagentní systém



## Abstract

This work deals with machine learning with symbolic representation of knowledge using Transparent intensional logic and its application on an element of a multi-agent system. Symbolic methods of machine learning are used as a tool for transforming experiences into formally written knowledge which are used for increasing intelligence of software applications which implements these methods. This work summarizes basics of machine learning, symbolic modes of machine learning, contains theory about Transparent intensional logic and syntax of its computer processable language TIL-Script. In the end analysis and design of an agent implementing symbolic method of machine learning using Transparent intensional logic is presented.



Key Words: machine learning, Transparent intensional logic, TIL-Script, symbolic methods, multi-agent system

# Obsah

# Seznam použitých zkratek a symbolů

| | | |
|---|---|---|
| TIL | - | Transparentní intenzionální logika |

# Seznam ilustrací a tabulek

# Úvod

V současnosti softwarové aplikace stále častěji nahrazují lidské pracovníky v činnosti tam, kde to před několika dekádami bylo naprosto nemyslitelné a v některých činnostech je dokonce předchází. S narůstající náročností úloh narůstají požadavky na jejich inteligenci a autonomii. Strojové učení hraje nemalou roli při navrhování inteligentních aplikací. Jedná se o široký obor informatiky umožňující rozšiřovat schopnosti inteligentních aplikací, a tím plnit komplexní úlohy nad rámec schopností navržených lidmi.

Jedním z oborů informatiky zabývající se inteligentními aplikacemi pro řešení komplexních úloh je obor multiagentních systémů. V tomto oboru se pomocí metod umělé inteligence, softwarového inženýrství a společenských věd, jako například ekonomie, vytváří flexibilní systémy s modulární konstrukcí obsahující autonomní prvky řešící daný problém. Multiagentní systémy využívají metod strojového učení za účelem zvýšení inteligence autonomních prvků, a tím zvýšení inteligence celého systému.

Strojové učení může v multiagentních systémech plnit hned několik rolí. V této práci je popsána role strojového učení jako nástroj pro získávání znalostí ze zkušeností pomocí symbolických metod. Symbolické metody strojového učení nejčastěji slouží pro získání obecných deskripcí popisující podmnožiny individuí nebo objektů. Tyto obecné popisy jsou dále v systému využity pro rozšíření jeho schopností, například pro identifikaci individua nebo objektu.

Aby mohly být znalosti systémem použity, musí být zapsány v jazyce, který je pro tento účel dostatečně expresivní, ale zároveň systémem zpracovatelný. Transparentní intenzionální logika (TIL) je formální jazyk natolik expresivní, že s ní lze analyzovat věty až na úrovni přirozeného jazyka. Použitím TIL získáme silný nástroj pro symbolický zápis znalostí získaných strojovým učením. Zápis znalostí v TIL je však pro zpracování softwarem nevhodný. Z tohoto důvodu byl vyvinut jazyk TIL-Script, pomocí kterého lze přepsat jakýkoliv zápis z TIL do jazyka zpracovatelného softwarem.

V případové studii, popsané v této práci, kombinuji symbolickou metodu strojového učení aplikovanou v autonomním prvku pro multiagentní systém, který pro zápis znalostí využívá expresivnost TIL zapsanou jazyce TIL-Script.

Strojové učení a jeho rozdělení podle základních charakteristik, konkrétně zpětné vazby a modelu funkce učení, je shrnuto v první kapitole. Kapitola druhá se věnuje rozšíření charakteristik strojového učení z kapitoly první o vlastnosti specifické pro multiagentní systémy. Symbolické modely strojového učení jsou popsány v kapitole třetí. Kapitola čtvrtá shrnuje teorii TIL, na kterou navazuje kapitola pátá popisující syntax jazyka TIL-Script. Případová studie agenta implementující symbolické strojové učení pomocí TIL je popsána v kapitole šesté.

# 1 Strojové učení

Strojové učení je podoblast umělé inteligence, která se zabývá algoritmy umožňujícími počítačovým programům zlepšit své chování na základě svých zkušeností. Zlepšením chování programu je myšleno rozšíření jeho chování, zpřesnění provádění chování nebo časové zefektivnění chování.

> ***Definice 1:*** *„Počítačový program se učí ze zkušeností E vzhledem k nějaké třídě úkolů T a míry výkonu P, pokud výkon na úkolu z T, měřený podle P se zlepšil se zkušeností E“ (Mitchell, 1997, s. 3 ) [1, s. 3]*

Například program, který se učí předpovídat počasí, může zlepšit svou předpověď měřenou v procentuální úspěšnosti předpovědi pomocí naměřených meteorologických dat.

Jedná se o široký obor obsahující různé přístupy k učení, metody nebo procesy využívající různé formy reprezentací znalostí pro učení. V této kapitole stručně jsou stručně popsány základní charakteristiky jednotlivých typů strojového učení. Nejdříve jsou popsány typy zpětných vazeb, se kterými program během učení pracuje, poté jsou popsány základní modely strojového učení.

## 1.1 Zpětná vazba

Podle Definice 1 potřebuje počítačový systém pro učení zkušenosti. Jedním z atributů těchto zkušeností je, jakou formu zpětné vazby, pokud vůbec nějakou, systému poskytují.

Pokud bychom chtěli charakterizovat metody strojového učení podle zpětné vazby, se kterou program pracuje, dostali bychom tři základní typy: *učení s učitelem* (suprvised machine learning), *učení bez učitele* (unsupervised machine learning) a *zpětnovazební učení* (reinforcement learning).

*Učení s učitelem* je metoda, kde program dostává ze zkušeností přímou zpětnou vazbu. Tyto zkušenosti jsou poskytnuty zdrojem, kterému se říká učitel. Učitel může být člověk, ale může se jednat i o jiný zdroj informací, například jiný program. V této metodě učení jsou zkušenosti popsány pomocí atributů rozdělených na vstupní a výstupní atributy. Hodnoty vstupních atributů popisují zkušenosti, hodnoty výstupních atributů jsou generovány pro program neznámou funkcí. Učitel trénuje program zkušenostmi, u kterých jsou poskytnuty hodnoty vstupních i výstupních atributů.

> ***Definice 2:*** *[2] Učení s učitelem je metoda strojového učení, ve které se program učí funkční závislost mezi hodnotami vstupních a výstupních atributů.*

Agent si pozorováním hodnot vstupních a výstupních atributů vytvoří svou vlastní funkci zvanou *hypotéza*, která aproximuje funkci neznámou (učenou).

Zkušenosti pro program učící se správnou předpověď počasí mohou být charakterizovány pomocí teploty vzduchu, vlhkosti vzduchu, rychlosti větru atd., tyto atributy označujeme jako vstupní. To, zda bude slunečno, zataženo nebo se třeba vyskytne bouře, je dáno právě hodnotami vstupních atributů, proto typ počasí je označen jako výstupní atribut. Zkušenosti poskytnuté učitelem, které by v hodnotách vstupních atributů obsahovaly meteorologická data typická pro krásné počasí, by v hodnotě výstupního atributu, *typ počasí*, měla hodnotu *slunečno.* Pozorováním hodnot vstupních a výstupních atributů u trénovacích dat si agent vytvoří hypotézu predikující typ počasí na základě hodnot vstupních atributů.

V dnešní době má strojové *učení s učitelem* v praxi širokou škálu uplatnění od vědy přes medicínu až po marketing nebo sociální sítě atd. Například v medicíně se hypotézy vzniklé strojovým učením používají pro klinickou diagnostiku, v rozpoznávacích procesech se využívají pro rozpoznávání hlasu, podpisů, obrazu, strojové učení se používá v marketingu pro propagaci produktů, v předpovědích počasí nebo v předpovědi provozu na komunikacích atd.

Více o strojovém učení s učitelem se dozvíte v [2], [3], [4], [1].

*Učení bez učitele* je metoda, ve které program ve zkušenostech nedostává žádnou zpětnou vazbu. Jejím cílem není predikce hodnoty výstupních atributů

> ***Definice 3:*** *Učení bez učitele je metoda strojového učení, ve které program pracuje s neklasifikovanými zkušenostmi, ve kterých hledá struktury nebo pravděpodobnostní rozdělení za účelem získání informací nebo redukce dat.*

U této metody jsou opět zkušenosti popsány pomocí atributů. Učitel však program trénuje zkušenostmi, u kterých nejsou poskytnuty výstupní atributy. V této metodě se nejčastěji používají dva přístupy. V prvním přístupu jsou data *shlukována do tříd* (hard clusterring). Při použití druhého typu metod se využívá metod statistiky a cílem je najít pravděpodobnostní rozdělení dat v třídách (soft clustering). Obě metody mají tedy podobnou funkci, a to shlukovat data za účelem porozumění nebo shlukovaní dat za účelem jejich redukce.

Shlukováním dat do tříd získáme společné atributy, které data v dané třídě sdílejí. Tyto společné atributy mohou být využity například pro porozumění datům. Při shlukování za účelem redukce dat se z popisů shluků vytváří prototypy, které představují zobecnění všech dat v dané třídě. Následné procesy pracují pouze s těmito prototypy namísto s větším množstvím dat.

Například program pro správu zvířat v zoologické zahradě může zvířata, která se vyznačují podobným chováním nebo podmínkami pro chov, shlukovat do tříd. Zvířecí druhy, které se nachází ve stejném shluku, mohou žít ve společných výbězích, takže pracovníci zoologické zahrady se můžou starat o více druhů zvířat stejným způsobem, a tím ušetří místo i čas.

V praxi se používá strojové učení bez učitele v segmentaci trhu se zákazníky, data miningu, analýze sociálních sítí, v biologii při tvorbě taxonomií zvířecích druhů, v psychologii a medicíně například pro shlukování různých druhů depresí, v klimatologii při hledání vzorců chování u atmosférického tlaku nebo teplotách oceánů nebo třeba v získávání informací z webu pomocí shlukování stránek atd.

Podrobněji se metodě shlukování věnují v [5], [3], [2], [6].

U *zpětnovazebního učení* program nedostává žádnou zpětnou vazbu přímo, ale prováděním akcí získává takzvané odměny nebo tresty.

> ***Definice 4:*** *Zpětnovazební učení je metoda, při které se program učí mapovat stavy na akce tak, aby maximalizoval odměnu za provedené akce.*

V mnoha komplexních doménách je *zpětnovazební učení* jediná vhodná metoda pro trénování agentů na vysoké úrovni, protože je často těžké pro člověka poskytnou přesnou a konzistentní zpětnou vazby, kterou agent potřebuje pro správné učení.

Program, učící se pomocí zpětnovazebního učení, nejčastěji nachází správnou sekvenci akcí metodou *pokus-omyl*, kde provádí sekvence akcí a zjišťuje, která přináší největší odměnu nebo pracuje s takzvanou *zpožděnou odměnou*, kde prováděním akcí program ovlivňuje odměny následujících stavů a tyto změny musí při učení zohlednit.

*Zpětnovazební učení* využívá Markovův rozhodovací proces[1] jako formální rámec pro interakci programu s nějakým prostředím na základě stavů prostředí, akcí a odměn.

Typickým problémem, který by program se *zpětnovazebním učením* mohl řešit je hra šach. Prostředí, ve kterém by se program pochyboval, by byla šachovnice a stav prostředí by udávaly aktuální pozice, na kterých se nachází šachové figurky. Každá šachová figurka má přesně definované tahy, které může provézt. Odměnu by program získal, pokud by šachovou partii vyhrál. Zkušenosti pro zlepšení chování programu (schopnost vyhrát) by program získával hraním šachových partií.

V praxi se zpětnovazební učení využívá například v chemii pro optimalizování chemických reakcí, v robotice například pro mapování video obrazu na akce robota, v řízení dopravy se učení využívá v multiagentních systémech, které řeší dopravní zácpy mezi křižovatkami, algoritmy učení řeší správu zdrojů v clusterech superpočítačů atd.

O zpětnovazebním učení se lze více dočíst v [7], [2], [3], [1].

## 1.2 Znalosti a modely strojového učení

Aby program mohl zlepšit výkon měřený na nějakém úkolu, potřebuje nějakým způsobem při učení přetransformovat zkušenosti na *znalosti*, které může využít při provádění daného úkolu, a tím zlepšit výkon.

> ***Definice 5:*** *Poznatek je popis vymezené části nějakého prostředí včetně zákonitostí, které v něm platí.*
>
> ***Definice 6:*** *Znalosti jsou vzájemně propojené struktury poznatků. Jsou tvořené popisem entit z daného prostředí (identifikace entit a jejich tříd), relacemi (vztahy mezi entitami) a procedurami (popis operací, které se mají během řešení daného problému provádět)*

*Znalosti* jsou informace o daném prostředí, které program využívá při řešení problému. Tyto *znalosti* program shromažďuje v *bázi znalostí*.

> ***Definice 7:*** *Báze znalostí je pasivní datová struktura neobsahující instrukce programu, které by měl program vykonávat, ale slouží pouze k parametrizování činností programu.*

Aby mohl program získat *bázi znalostí*, potřebuje funkci, která vytvoří ze zkušeností *znalosti* potřebné k řešení daného problému. Tato funkce bude představovat učící se mechanismus. Historickým vývojem vznikly různé přístupy k výběru typy této funkce. Mezi nejhlavnější přístupy reprezentace funkce (modely) učení patří *metody umělých neuronových sítí*, *metody genetických algoritmů*, *stochastické metody* a *metody symbolické reprezentace*.

[1] V Markově rozhodovacím procesu zná na začátku učení program pouze množinu možných stavů prostředí a množinu možných stavů. Přechodovou funkci mezi stavy ani odměňovací funkci program nezná. Po každé akci program pozoruje nový stav prostředí a získává odměnu.

U strojového učení *pomocí umělých neuronových sítí*, známého taky jako paralelní distribuované procesy, pochází inspirace z funkce biologických neuronů v mozku zvířat.

> ***Definice 8:*** *Umělá neuronová síť je struktura tvořená samostatnými uzly zvanými neurony. Každý neuron může mít několik vstupů a produkuje jedinou výstupní hodnotu. Hodnota výstupu je generována aktivační funkcí neuronu z váženého součtu hodnot na jeho vstupech a vah přiřazených jednotlivým vstupům. Neurony jsou často strukturovány do vrstev tvořící vícevrstvou neuronovou síť, kde vstupem neuronu je výstup neuronu z předcházející vrstvy.*

Existuje několik typů *umělých neuronových sítí* a několik typů neuronů. Mezi základní typy sítí patří takzvaná *feed-foward* neuronová síť, kde neurony propagují své hodnoty pouze do následující vrstvy oproti rekurentním sítím, které vrací výstup zpět do svých vstupů. Neurony se mohou odlišovat typem aktivační funkce, například funkce jednotkového skoku, lineární funkce, hyperbolický tangent atd.

Při učení *umělé neuronové sítě* je cílem takové nastavení vah, aby vytvářely správnou hodnoty výstupních hodnot pro daný vstup. Při učení s učitelem se porovnává výstup tvořený sítí a výstup daný učitelem. Váhy se následně upravují tak, aby se snížil rozdíl mezi tvořeným a požadovaným výstupem. Při učení bez učitele je snaha upravit váhy v síti tak, aby poskytovaly konzistentní výstupní hodnoty při poskytnutí různých vstupů pocházejících ze stejné třídy.

Více se o neuronových sítí dočtete v [8], [9], [2], [3], [4].

Strojové učení s *metodou genetických algoritmů* bylo inspirováno biologickou evolucí.

> ***Definice 9:*** *„Genetický algoritmus je heuristické hledaní, které napodobuje proces přirozené evoluce za účelem vytvoření užitečného řešení pro optimalizační a vyhledávací problémy. Genetické algoritmy jsou podmnožinou evolučních algoritmů, které řeší optimalizační problémy použitím technik inspirovaných přirozenou evolucí, jako dědičnost, mutace, selekce a křížení.“ [10, s. 3]*

*Genetické algoritmy* jsou aplikovány na jedince v populaci. Jedinci jsou popsáni takzvaným *chromozomem* a *fitness hodnotou*.

*Chromozom* je řetězec informací popisující chování a atributy jedinců. Nejčastěji je *chromozom* popsán řetězcem jedniček a nul, ale může být popsán i jinými způsoby, jako jsou reálná čísla, matice nebo vektory, atd. *Chromozomy* jsou dále dělené na jednotlivé geny, což jsou v rámci algoritmu více nedělitelné části.

Kvalita jedinců je hodnocena *fitness hodnotou*. Pro každý problém řešený pomocí genetických algoritmů je nutno vytvořit fitness funkci, která bude adekvátně k problému přiřazovat jedincům v populaci *fitness hodnotu*.

*Genetické algoritmy* nejčastěji používají operace selekce, mutace a klížení na populaci. Výsledkem těchto operací na populaci je nová populace obohacená o nové jedince vzniklé prováděním zmíněných operací. Operace se aplikují do té doby, dokud se v populaci neobjeví jedinci s požadovanými atributy.

Ve strojovém učení jedinci představují hypotézy. Například ve strojovém *učení s učitelem* by hypotézy v populaci predikovaly hodnoty výstupních atributů na příkladech poskytnutých učitelem. Fitness

funkce by udávala přesnost predikované hodnoty výstupního atributu hodnoty od hodnoty dané učitelem.

*Genetické algoritmy* se používají například v automobilovém průmyslu pro návrhy kompozitních materiálů nebo návrhy aerodynamických tvarů, v robotice pro řešení optimalizace návrhů robotů, v telekomunikacích pro optimalizaci návrhů komunikačních sítí atd.

O metodách *genetických algoritmů* se více dočtete v [11], [10], [1], [4].

U *stochastického*[2] *přístupu* využívají modely v mechanismu učení pravděpodobnost, konkrétně Bayesův teorém. V tomto přístupu se na data díváme jako na instance nějakých náhodných proměnných popisujících nějaký svět a hypotézy jsou pravděpodobnostní teorie popisující svět.

> ***Definice 10:*** *Bayesův teorém nám poskytuje možnost, jak vypočítat podmíněnou pravděpodobnost* $P(h|D)$ *z pravděpodobnosti* $P(h)$ *,* $P(D)$ *a* $P(D|h)$ *, kde* $P(h|D)$ *je pravděpodobnost, že hypotéza h je správná vzhledem k pozorovaným datům D,* $P(h)$ *je apriorní pravděpodobnost hypotézy*[3] $h$*,* $P(D)$ *je apriorní pravděpodobnost dat*[4] *a* $P(D|h)$ *je pravděpodobnost, že data D budou pozorována v nějakém světě, kde hypotéza* $h$ *je správná. Bayesův teorém je zapsán:*
>
> $$P(h|D) = \frac{P(D|h)\,P(h)}{P(D)}$$

Mezi nejznámější modely stochastického přístupu patří MAP model a Bayesovské sítě

> ***Definice 11:*** *MAP model (maximum a posteriori probability) je model strojového učení hledající hypotézu s maximální pravděpodobností* $P(h|D)$.

> ***Definice 12:*** *Bayesovská síť je grafový model tvořený systémem podmíněných pravděpodobnostních distribucí a acyklickým grafem, jehož uzlům jsou přiřazeny náhodné veličiny.*

Modely *stochastického přístupu* mají v praxi široké uplatnění. Ve zpracování obrazu se používají například pro odstranění šumu v obrazu, v lingvistice pro automatické překlady cizích jazyků, v analýze zvuku pro vzorkování signálu nebo pro rozpoznávání hlasu, v komputační biologii pro analýzu vývoje DNA nebo v medicíně pro klinickou diagnostiku.

Více se o stochastických metodách dočtete v [12], [1], [2], [3].

Čtvrtým základním modelem pro reprezentování mechanismu učení jsou takzvané *metody symbolické reprezentace*. Tyto metody jsou založené podle [4] na předpokladu, že hlavní vliv na chování programu má kolekce explicitně reprezentovaných znalostí o řešeném problému.

---

[2] Stochastika je matematický obor, který se zabývá zkoumáním a modelováním náhodných jevů. Jedná se o souhrnný název pro teorii pravděpodobnosti a matematickou statistiku.

[3] Apriorní pravděpodobnost hypotézy je výchozí pravděpodobnost každé hypotézy, že je správná, aniž bychom pozorovali jakékoliv data.

[4] Apriorní pravděpodobnost dat je výchozí pravděpodobnost, že data budou pozorována.

***Definice 13:*** *Učení pomocí metod symbolické reprezentace je vytváření nebo modifikování explicitně reprezentovaných znalostí o problému řešeném programem.*

*Explicitní reprezentace* znalostí se nejčastěji objevuje ve formě symbolických struktur nějakého formálního jazyka.

***Definice 14:*** *Formální jazyk L nad abecedou Σ je podmnožina Σ*, tedy libovolná množina slov nad touto abecedou.*

***Definice 15:*** *Abeceda je libovolná konečná množina značená symbolem Σ. Prvky jsou nazývány symboly abecedy.*

***Definice 16:*** *Slovo (řetězec) v nad abecedou Σ je libovolná konečná posloupnost znaků této abecedy tvořená formální gramatikou.*

***Definice 17:*** *Formální gramatika je množina pravidel pro tvorbu slov formálního jazyka L nad danou abecedou.*

Jedním z nejpoužívanějších *formálních jazyků* pro reprezentaci znalostí se ve strojovém učení používá jazyk *formální logiky*.

***Definice 18:*** *„Formální logika definuje a studuje abstraktní odvozovací pravidla (tj. „formy úsudků"), jejichž platnost nezávisí na významu pojmů, které v nich vystupují." [13, s. 2]*

Volba typu *formální logiky* závisí na typu *znalostí*, které program potřebuje pro řešení problému. Pokud jsou například tyto *znalosti* typu propozičně-logických teorií, nejvhodnějším formálním jazykem pro jejich reprezentaci by byl jazyk výrokové logiky. Kdyby *znalosti* potřebné pro řešení problému byly relačně-logické teorie, pro jejich reprezentaci bychom zvolili jazyk predikátové logiky 1. řádu.

Použitím symbolického zápisu ve *formálním jazyce* dostaneme při popisu *znalostí* o problému, který program řeší, *výroky*.

***Definice 19:*** *[2] Výrok je věta zapsaná v jazyce, který má v daném světě pravdivostní hodnotu.*

Pravdivostní hodnota *výroku* může nabývat hodnot *Pravda* nebo *Nepravda.* Výhoda zápisu znalostí pomocí symbolické reprezentace je pro program jednodušší než uchovávat informace například ve vyhledávací tabulce[5]. V [2] se dočteme, že symbolická reprezentace nám dává výhodu přirozenějšího zápisu *znalostí*, který je do jisté míry i dobře čitelný pro člověka. Reprezentace *znalostí výroky* je modulární a můžeme si vybírat z mnoha typů formálních zápisů, jako již zmíněný jazyk výrokové logiky, predikátové logiky atd.

Další výhodou symbolické reprezentace *znalostí* je, že program může uchovávat ve své vnitřní reprezentaci model relačně-logické teorie o individuích, aniž by znal všechna individua, která se v dané doméně problému nacházejí nebo věděl, kolik jich je. Pokud program nezná všechna individua, znamená to, že nezná ani všechny vlastnosti těchto individuí. Program tyto chybějící *znalosti* může doplnit do své reprezentace prostředí, až je objeví na základě nějaké zkušenosti.

[5] Například pokud program uchovává znalosti o nějakém prostředí čítající 25 propozic, pak tato reprezentace může popsat 33 554 432 možných stavů prostředí.

Operace vytváření a modifikování reprezentovaných *znalostí* se řídí formou *předpojatosti* implementovanou modelem učení.

> ***Definice 20:*** *Předpojatost je upřednostňování jedné reprezentace znalostí před jinou.*

V *symbolických metodách strojového učení* se se uplatňuje takzvaná *jazyková předpojatost*, která klade omezení na *formální jazyk* použitý pro symbolický zápis *znalostí*. V [4] se uvádí, že mezi *jazykové předpojatosti* například patří *konjunktivní předpojatost*, *předpojatost pomocí limitovaného počtu disjunktů*, *předpojatost zápisem pomocí hornových klauzulí*.

> ***Definice 21:*** *Konjunktivní předpojatost klade omezení na formální jazyk, takové, že znalosti mohou být symbolicky zapsané pouze jako konjunkce literálů.*

> ***Definice 22:*** *Předpojatost limitováním počtu disjunktů v symbolickém zápisu znalosti znamená zavedení malého počtu disjunkcí do konjunktivního zápisu znalosti pro rozšíření expresivity znalosti.*

> ***Definice 23:*** *Hornova klauzule je klauzule obsahující nejvýše jeden pozitivní literál.*

Strojové učení využívající metody symbolické reprezentace se v praxi používá například pro konceptuální učení, zpracování přirozeného jazyka, induktivní logické programování, vytváření znalostních bází použité pro klasifikaci dat, atd.

Strojové učení pomocí symbolické reprezentace je dále popsáno v [4], [14], [1], [2], [3].

# 2 Učení v multiagentních systémech

Tato kapitola je rozšířením Kapitoly 1 charakterizující učení v multiagentních systémech. Nejdříve je stručně popsáno, co je agent a multiagentní systémy, následně jsou popsány důvody, proč je učení v těchto systémech charakteristické. Na závěr jsou popsány základní charakteristiky učení v multiagentních systémech.

> ***Definice 24:*** *„Agent je entita zkonstruována za účelem kontinuálně a do jisté míry autonomně plnit své cíle v adekvátním prostředí na základě vnímání prostřednictvím senzorů a prováděním akcí prostřednictvím aktuátorů. Agent přitom ovlivňuje podmínky v prostředí tak, aby se přibližoval k plnění cílů.“ [15, s. 12]*
>
> ***Definice 25:*** *Multiagentní systém je systém nezávislých agentů, kteří se snaží splnit určitý cíl pomocí kooperace nebo soutěžení mezi sebou.*

V *multiagentních systémech* máme 2 základní důvody, proč zkoumat strojové učení z jiných pohledů než jako učení u klasických programů implementující strojové učení.

První důvodem je prostředí, ve kterém *agenti* pracují. Tato prostředí jsou často natolik komplexní, nepředvídatelné, obsáhlé a dynamické, že designéři *agentů* nejsou schopni *agenty* navrhnout tak, aby jejich *znalosti* a schopnosti, poskytnuté designérem, byly pro dosažení jejich cílů vždy dostatečné. Jedinou možností tedy je zakomponovat schopnost učení se, aby *agent* v závislosti na svých zkušenostech zvýšil výkon prováděných akcí, a tím dokázal dojít ke svému cíli.

Druhým důvodem jsou charakteristiky speciální pro doménu *multiagentních systémů*. V takovém systému *agenti* nemusí pracovat jako samostatné programy, ale jejich akce mohou být ovlivněny ostatními *agenty* (zdržení agentem, urychlení provedení akce jiným agentem, umožnění provedení nějaké akce díky jinému agentovi atd.). Stejně to platí i pro jejich učení. V jednom *multiagentním systému* můžou být *agenti*, kteří se liší od sebe v metodě nebo přístupu učením, a tím se vzájemně ovlivňovat.

V [16] se dozvíme, že z těchto dvou důvodů je náhled na učení rozšířen o charakteristiky, které u samostatných systémů nemá smysl řešit. Dvě základní obecné kategorie učení v *multiagentních systémech* jsou:

- Distribuované učení – procesu učení se účastní dva a více agentů, části procesu mohou samostatně vykonávat různí agenti
- Centralizované učení – model učení je ve všech částech procesu vykonáván jediným agentem a proces učení není ovlivněn dalšími agenty

Učení v *multiagentních systémech* se dá dále charakterizovat pomocí několika dalších atributů například:

- Úroveň decentralizace učení – úroveň specifikující, do jaké míry je učení v systému centralizováno nebo distribuováno (extrémní případ centralizovaného učení by byl, kdyby v celém systému pouze jeden agent implementoval proces učení pro ostatní agenty v systému,

druhý extrém by představoval systém, ve kterém všichni agenti paralelně pracují na jednom procesu učení)

- Úroveň interakce mezi agenty při učení – atribut zahrnující další atributy jako například úroveň interakce (specifikace typu interakce), délka trvání interakce , frekvence interakce, vzory v interakci nebo variabilita v interakci
- Úroveň zapojení agenta v procesu učení – atribut specifikující zapojení agenta do procesu, například jeho role v procesu učení

Další charakteristiky učení jako modely učení nebo typ zpětné vazby při učení jsou shodné jako u učení samostatných programů zmíněných v Kapitole 1.

Více o *multiagentních systémech* se můžete dočíst například v [15], [16], [2].

# 3 Modely strojového učení se symbolickou reprezentací

V této kapitole jsou popsány nejznámější modely strojového učení využívající symbolickou reprezentaci znalostí. Kapitola je strukturované podle charakteristik zmíněných v kapitole 1.1. Nejprve jsou popsány modely strojového učení se supervizorem, konkrétně modely konceptuálního učení hledající nejvhodnější hypotézu konceptu a rozhodovací stromy, dále je popsán model strojového učení bez učitele, konkrétně konceptuální shlukování.

## 3.1 Konceptuální učení

Podle [1] je jedním z nejčastějších úkolů strojového učení se symbolickou reprezentací znalostí získání obecného *konceptu* ze specifických zkušeností. Zkušenosti v konceptuálním učení jsou popisy příkladů (instancí) daného konceptu.

> ***Definice 26:*** *[1] Koncept je popis podmnožiny objektů nebo událostí nad nějakou větší množinou nebo jako funkční závislost nad danou množinou*[6].

> ***Definice 27:*** *[1] Konceptuální učení je odvození funkce s oborem hodnot {Pravda, Nepravda} ze vstupních atributů a výstupních atributů trénovacích příkladů.*

Příkladem *konceptu* pro *konceptuální učení* je například koncept *pták* (podmnožina zvířat), *strom* (podmnožina rostlin), *slunečný den* (podmnožina typů počasí) atd. *Konceptuální učení* se nejčastěji implementuje pomocí metody strojového *učení s učitelem*, kdy učitel poskytuje programu, který se učí nějaký *koncept*, množinu příkladů (pozitivní instancí) konceptu a agent si pozorováním těchto příkladů vybuduje hypotézu.

Hypotéza *pták* by mohla vypadat v symbolické reprezentaci pomocí Predikátové logiky 1. řádu takto:

$$Počet\_nohou(x, 2) \wedge Teplokrevný(x) \wedge Vejce\_snášející(x) \wedge Obratlovec(x) \wedge \ldots \\ \wedge Opeřený(x) \supset Pták(x)$$

Takováto hypotéza určí pro každé individuum $x$ z univerza zvířat pravdivostní hodnotu. Pokud pravdivostní hodnota nabyde hodnoty *Pravda*, znamená to, že individuum je pták, pokud pravdivostní hodnota bude *Nepravda*, individuum $x$ pták není. Aby individuum bylo pták, musí splňovat podmínky daného konceptu (jeho atributy), zapsané v hypotéze pomocí predikátů (predikátových znaků), například predikát $Počet\_nohou(x, 2)$ symbolizuje atribut, že individuum má dvě nohy, což je atribut každého ptáka (neuvažujeme zde o případech zvířat, které nějakým způsobem přišly o nohu nebo se například narodily bez nohou).

Při učení tohoto konceptu by učitel poskytnul agentovi množinu příkladů (instancí) daného konceptu. Příklady by byly popsané pomocí hodnot vstupních a výstupních atributů. Hodnoty vstupních atributů popisují daný příklad (stejně jako výše zmíněné predikáty hypotézy), hodnota výstupního atributu

[6] Funkční závislost, která každému objektu z množiny objektů určí, zda patří do konceptu nebo ne. Funkční závislost (konceptu *pták*) definována nad množinou zvířat by vracela pro ptáky pravda a pro ostatní zvířata nepravda.

(predikát *Pták*) popisuje, zda se jedná o pozitivní nebo negativní příklad. Agent si pozorováním závislostí mezi hodnotami vstupních a výstupních příkladů vytvoří vlastní funkční závislost (hypotézu) konceptu.

### 3.1.1 Učení jako hledání hypotézy

Aby mohl program získat *hypotézu* z příkladů poskytnutých učitelem, potřebuje množinu operací, které jsou schopné manipulovat s reprezentací *hypotézy*. Při použití metod symbolické reprezentace *znalostí* se nejčastěji používají operace generalizace a specializace hodnot atributů zapsaných pomocí symbolů v reprezentaci[7].

V [2] definují konceptuální prostor (prostor hypotéz), ve kterém se nacházejí všechny potencionální *hypotézy* daného *konceptu*, formou reprezentace a operacemi manipulující s reprezentací. Úkolem v *konceptuálním učení* je hledání *hypotéz*, které jsou *konzistentní* na datech.

> ***Definice 28:*** *Konzistentní hypotéza je taková hypotéza, která pro každý příklad predikuje správnou hodnotu výstupního atributu, tzn.* $\forall x \in P, h(x) = \gamma(x)$*, kde P je množina příkladů poskytnutých učitelem, h je hypotéza predikující hodnoty výstupního atributu příkladu,* $\gamma$ *je neznámá funkce generující hodnoty výstupních atributů pro každé x v P.*

Daný *koncept* může mít potencionálně nekonečno příkladů, které mají potencionálně nekonečno atributů. Jelikož má program informace o daném *konceptu* pouze z příkladů poskytnutých učitelem, musí provádět heuristické hledání[8] za cílem najít efektivně všechny potencionální konzistentní hypotézy.

> ***Definice 29:*** *Induktivní učení je hledání hypotézy z omezeného příkladů, která je konzistentní i na doposud neviděných příkladech daného konceptu.*

Pro použití metod *induktivního učení* musíme splnit tři předpoklady:

- Koncept má pouze jeden výstupní atribut Y (Boolovský).
- Hypotéza jasně predikuje hodnotu výstupního atributu Y ( na rozdíl od pravděpodobnostní predikce).
- V datech se nenachází šum[9].

### 3.1.2 Uspořádání hypotéz

V [1] využívají pro efektivní hledání ve velkých i potencionálně nekonečných prostorech hypotéz, algoritmy konceptuálního učení využívající uspořádání hypotéz v prostoru hypotéz od nejspecifičtější po nejobecnější. Využitím tohoto uspořádání lze prohledávat prostor hypotéz, aniž bychom museli zpracovávat jednu hypotézu po druhé.

---

[7] Při použití modelu učení pomocí umělých neuronových sítí by to byly operace upravující váhy v neuronech.
[8] Heuristické hledání v hledání hypotéz je soubor takový pravidel pro výběr hypotézy, který vede hledání k nalezení hypotézy, která bude konzistentní na všech příkladech daného konceptu.
[9] Šum v datech představují například neadekvátní atributy pro správnou predikci hodnoty výstupního atributu příkladu, chybějící data, chyby v hodnotách atributů atd.

Mějme příklad, kdy pracujeme s *konceptem auta* skládajícího se ze vstupních atributů: *počet dveří*, *barva*, *délka auta v metrech*, *stupeň výbavy*, *typ paliva*. Mějme dvě hypotézy $h_1$, $h_2$ predikující hodnoty výstupních atributů (zda se jedná o auto či ne) pro příklady poskytnuté učitelem. Pro jednoduchost hypotézu symbolicky reprezentujeme jako množinu prvků reprezentující hodnoty vstupních atributů, kde prvek množiny reprezentovaný symbolem *X* reprezentuje jakoukoliv hodnotu akceptovatelnou atributem nebo prvek množiny reprezentuje konkrétní hodnotu atributu.

$$koncept\ auta = \{počet\ dveří, barva, délka\ auta\ v\ metrech, stupeň\ výbavy, typ\ paliva\}$$

$$h_1 = \{X, modrá, X, X, benzín\}$$

$$h_2 = \{3, modrá, X, základní, benzín\}$$

Pokud vezmeme množiny příkladů, které jsou pozitivně identifikovány jako instance *konceptu auta* hypotézami $h_1$ a $h_2$, zjistíme, že množina identifikována hypotézou $h_1$ je mohutnější než množina identifikována hypotézou $h_2$, protože $h_1$ klade na příklady méně podmínek/omezení (je benevolentnější). Dalo by se intuitivně říct, že hypotéza $h_1$ je obecnější než hypotéza $h_2$.

Obecně je tato relace definována takto: Pro jakýkoliv příklad $p$ z prostoru příkladů $P$ a hypotézu $h$ z prostoru hypotéze platí, že $p$ "splňuje" (odpovídá hypotéze) $h$ právě tehdy, když $h(x) = Pravda$. Relace *Být stejně nebo více obecný* mezi hypotézami je definována pomocí množiny příkladů, které "splňují" hypotézy $h_i$, $h_k$ (odpovídají hypotézám).

> ***Definice 30:*** *[2] Hypotéza $h_i$ je Obecnější nebo stejně obecná než hypotéza $h_k$ právě tehdy, když hypotéza $h_k$ implikuje hypotézu $h_i$.*

V [1] definují relaci následovně: Nechť $h_i$ a $h_k$ jsou hypotézy s jedním boolovským výstupním atributem definované nad množinou příkladů P, pak $h_i$ je *Obecnější nebo stejně obecná* než hypotéza $h_k$ ($h_i \geq h_k$) právě tehdy, když:

$$\forall p \in P[h_k(p) = Pravda \supset h_i(p) = Pravda]$$

Jinými slovy hypotéza $h_i$ je *Obecnější nebo stejně obecná* než hypotéza $h_k$ právě tehdy, když příklady, které "splňují" $h_k$, také "splňují" $h_i$.

Relace *Obecnější nebo stejně obecná* tvoří relaci částečného uspořádání v prostoru hypotéz, tzn. reflexivní antisymetrická a reflexivní .

Mějme tři hypotézy $h_1$, $h_2$, $h_3$ z prostoru hypotéz, které jsou symbolicky reprezentovány stejně jako příklad výše pomocí množiny hodnot vstupních atributů popisující koncept auta:

$$h_1 = \{X, X, X, X, benzín\}$$

$$h_2 = \{X, modrá, X, základní, benzín\}$$

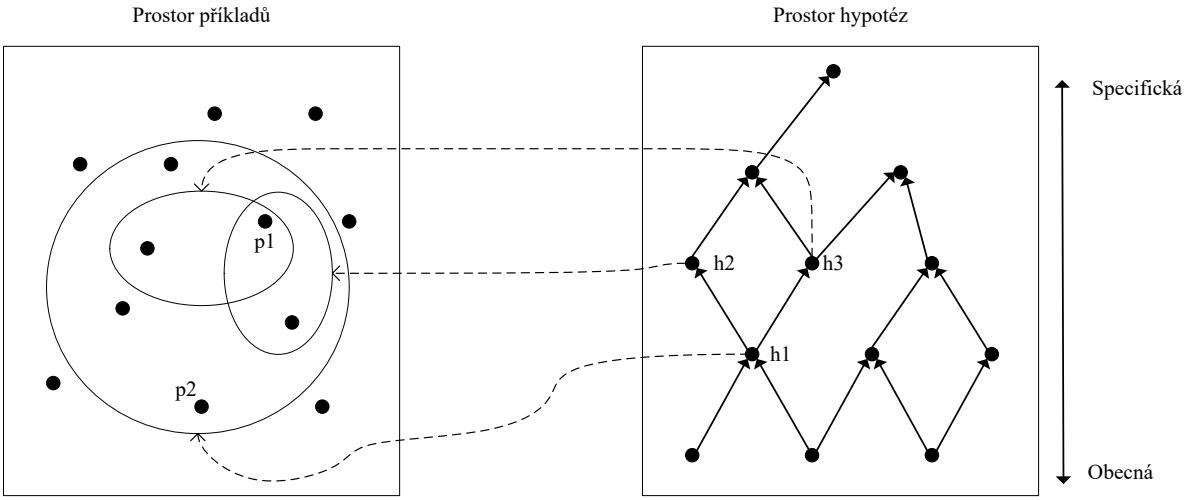
$$h_3 = \{3, X, 3, X, benzín\}$$

A příklady $p_1$, $p_2$, které jsou symbolicky popsané stejným způsobem jako hypotézy:

$$p_1 = \{4, bílá, 3, střední, benzín\}$$

$$p_2 = \{3, modrá, 3, základní, benzín\}$$

Na Obrázku 1 vidíme v levém obdélníku prostor příkladů, ve kterém jsou elipsami označeny podmnožiny definované hypotézami, nacházející se v obdélníku napravo, představující prostor hypotéz. Každá hypotéza definuje podmnožinu příkladů, které "splňují" danou hypotézu. Plně vybarvené šipky v obdélníku napravo představují relace *Obecnější nebo stejně obecná* mezi hypotézami v prostoru hypotéz. Jak lze z obrázku vyčíst, množina, která je definována hypotézou $h_1$, obsahuje v sobě prvky, které se nachází v podmnožinách definovaných hypotézami $h_2$ a $h_3$. To znamená, že hypotéza $h_1$ je obecnější než hypotézy $h_2$ a $h_3$. Podle definice $h_2(p1) = True \supset h_1(p1) = True$. Jinými slovy příklady, které odpovídají konceptům popsaných hypotézami $h_2$ a $h_3$, budou zároveň odpovídat obecnějšímu konceptu popsaným hypotézou $h_1$.




Obrázek 1: Příklady a hypotézy v částečném uspořádání

Duálně k relaci *Obecnější nebo stejně obecná* existuje relace *Specifičtější nebo stejně specifická* ($h_k \leq h_i$).

> ***Definice 31:*** *Hypotéza $h_k$ je specifičtější nebo stejně specifická než $h_i$ právě, když je $h_i$ je obecnější nebo stejně obecná jako $h_k$.*

V příkladě výše je hypotéza $h_2$ *specifičtější* než hypotéza $h_1$.

V následující kapitole uvedu heuristické hledání, které efektivně využívá strukturu zavedenou ≥ relací pro *konceptuální učení*.

### 3.1.3 Prohledávání Version space

V [4] se dočteme, že pro to, aby se program mohl naučit hypotézu z příkladů, potřebuje sadu operátorů, které manipulují s reprezentací znalostí. Jelikož je v prostoru hypotéz zanesena struktura obecnosti (specifičnosti) mezi možnými hypotézami, jsou dvě hlavní operace pro manipulaci s reprezentací generalizace (zobecňování) a specializace (upřesňování). Generalizace nejčastěji obsahuje tyto pod-operace:

- Nahrazení konstanty proměnnou v atributu, například:

$$[\ldots \wedge\ barva(p, modrá) \wedge \ldots\ \supset Auto(p)]$$

  za konstantu *modrá* dosadíme proměnnou *X* reprezentující jakoukoliv hodnotu akceptovanou atributem:

$$[\ldots \wedge\ barva(p, X) \wedge \ldots\ \supset Auto(p)]$$

- Odstranění atributu z výroku, například:

$$[\ldots \wedge\ barva(p, modrá) \wedge výbava(p, základní) \wedge \ldots\ \supset Auto(p)]$$

  Odstraněním atributu *výbava* budou hypotézu "splňovat" příklady, které by u původní hypotézy tuto podmínku nesplňovaly.

$$[\ldots \wedge\ barva(p, modrá) \wedge \ldots\ \supset Auto(p)]$$

- Přidání disjunkce do propozice, například :

$$[\ldots \wedge\ barva(p, modrá) \wedge \ldots\ \supset Auto(p)]$$

  Přidáním disjunkce mezi dva shodné atributy docílíme splnění podmínky barvy pro dvě hodnoty:

$$[\ldots \wedge\ [barva(p, modrá) \vee barva(p, červená)]\ \wedge \ldots\ \supset Auto(p)]$$

- Nahrazením hodnoty atributu jeho obecnější třídou, například:

$$[\ldots \wedge\ počet\_kol(p, 4) \wedge \ldots\ \supset Nákladní\_auto(p)]$$

  Hodnotu atributu *počet_kol* nahradíme obecnější třídou *sudá čísla*:

$$[\ldots \wedge\ počet\_kol(p, sudý) \wedge \ldots\ \supset Nákladní\_auto(p)]$$

Duálně k pod-operacím generalizace existují pod-operace specializace, které specializují hypotézu pro vyloučení určitých příkladů.

Aplikacím těchto operací (generalizace a specializace) se nad prostorem hypotéz, podle příkladů předložených učitelem, vytváří tzv. *Version space*.

> ***Definice 32:*** *Version space je podmnožina hypotéz, která je konzistentní na dosud viděných příkladech.*

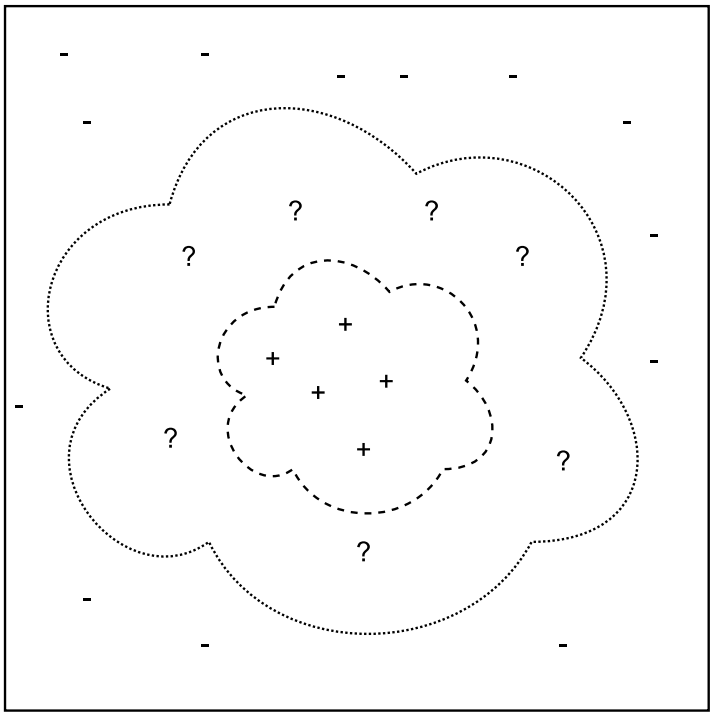
Příklady předložené učitelem podle [4] mohou být pozitivní instance konceptu nebo negativní instance konceptu. Hledanou hypotézu lze získat pouze z pozitivních příkladů pomocí generalizace. U tohoto způsobu je však důležité, aby učitel vybíral příklady tak, aby během procesu učení nezískal příliš obecnou hypotézu, tedy takovou hypotézu, která by pokryla i příklady, které nejsou instancí konceptu. Takový problém se nazývá *over-generalizace* hypotézy. Aby nedošlo k *over-generalizaci* hypotézy, může učitel použít při učení negativní příklady (negativní instance konceptu) , pomocí kterých hypotézu

specializuje. Použitím negativních příkladů specializujeme hypotézu, a tím odstraníme z *Version space* příliš obecné hypotézy.

V [2] definují *Version space* jako množinu hypotéz konzistentních na datech vymezených pomocí *obecné* a *specifické* hranice.

> ***Definice 33:*** *Obecná hranice je taková množina hypotéz, že ve Version space se nenachází hypotéza, která by byla obecnější než hypotézy v obecné hranici.*
>
> ***Definice 34:*** *Specifická hranice je množina takových hypotéz, že ve Version space se nenachází hypotéza, která by byla specifičtější než hypotézy ze specifické hranice.*

Vymezení *Version space* pomocí těchto dvou hranic je znázorněna na Obrázku 2. *Specifická hranice* je označena tečkovanou čarou, *obecná hranice* čárkovanou čarou. Symbolem – jsou označeny hypotézy příliš specifické pro *koncept*, symbolem + jsou označeny příliš obecné hypotézy. Otazníkem jsou označeny *hypotézy konzistentní* na dosud viděných příkladech.



Obrázek 2: Znázornění Version space

## 3.1.4 Hledání od obecnějšího ke specifickému

Prvním typem hledání hypotézy je prohledávání prostoru hypotéz od nejobecnější hypotézy, kterou "splňují" všechny objekty v prostoru příkladů, i ty, které ve skutečnosti neodpovídají hledanému konceptu, po nejspecifičtější možnou hypotézu, která popisuje námi hledaný koncept.

Při hledání podle [4] budeme operacemi specializace, vzhledem k vybraným příkladům *konceptu*, specializovat *hypotézu* tak, abychom získali *maximálně obecné hypotézy*, které však nebudou pokrývat příklady, které instancemi *konceptu* nejsou.

***Definice 35:*** *Hypotéza $h$ je maximálně obecná tehdy, pokud pouze instance konceptu tuto hypotézu "splňují", nikoliv však příklady, které instancemi konceptu nejsou. Pro každou další hypotézu $h'$ z Version space konceptu platí, že je $h \geq h'$, čili $h$ je obecnější nebo stejně obecná než $h'$.*

Algoritmus učení odpovídající hledání hypotéz od nejobecnější po nejspecifičtější je specifikován následovně:

Vstup:
Es… množina pozitivních a negativních příkladů konceptu

Výstup:
G…množina maximálně obecných hypotéz konceptu

Algoritmus:
BEGIN
Do G přiřaď nejobecnější hypotézu konceptu;
P obsahuje všechny použité pozitivní příklady;
FOR EACH pozitivní příklad p ∈ ES
    BEGIN
    Smaž z G všechny hypotézy, které nepopisují p;
    Přidej p do P;
    END
FOR EACH negativní příklad n ∈ ES
    BEGIN
    FOR EACH g ∈ G, které popisuje n nahraď g nejobecnější specializací, která nepopisuje n;
    Odstraň z G všechny specifičtější hypotézy než jiné hypotézy v G;
    Odstraň z G všechny hypotézy, které nepopisují nějaký příklad z P;
    END
END

Učení pomocí hledání hypotézy od *nejobecnější* po *nejspecifičtější hypotézu* je popsáno na příkladu programu, který se učí *koncept auta*, které jako palivo používá CNG (nezáleží nám na velikosti auta ani na barvě). Pro jednoduchost budeme pracovat na omezeném prostoru příkladů, kde příklady jsou popsány pouze těmito vstupními atributy a jejich hodnotami:

- Typ paliva: {benzín, nafta, CNG}
- Barva: {zelená, bílá, černá}
- Velikost: {malá, velká}

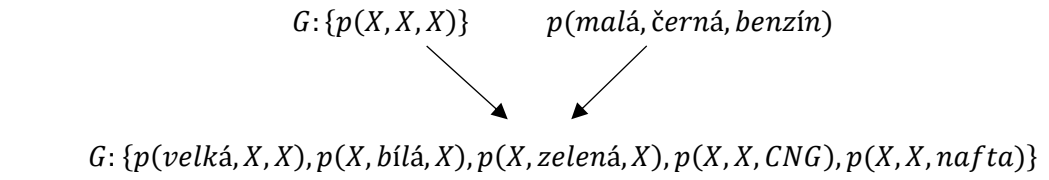
Hodnota atributu *X* bude opět představovat libovolnou hodnotu akceptovatelnou atributem.

Příklady jsou symbolicky popsány jako množina hodnot atributů definující příklad $p(Typ\ paliva, Barva, Velikost)$.

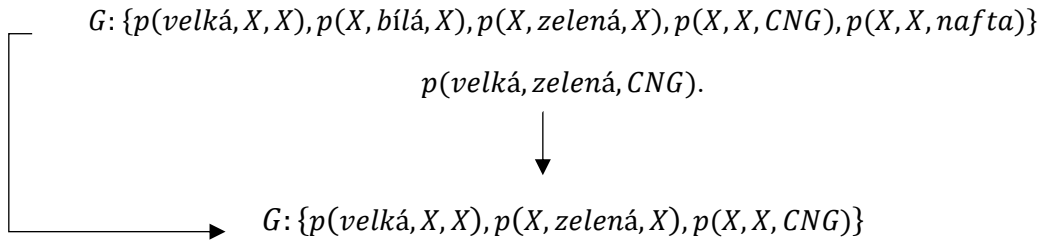
Podle výše zmíněného algoritmu se nejdříve do *G* nastaví neobecnější hypotéza:

$$G: \{p(X, X, X)\}$$

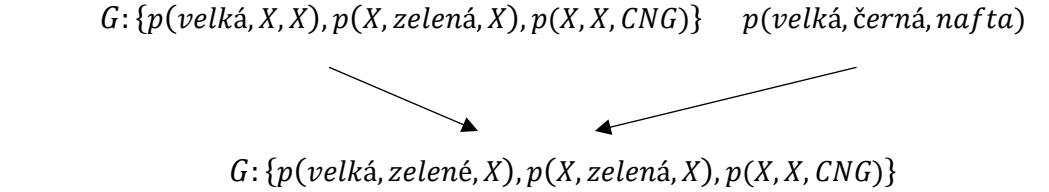
Tuto hypotézu "splňují" všechny příklady, protože akceptuje jakoukoliv hodnotu atributu. Nyní program zpracuje první negativní příklad představující malé, černé auto jezdící na benzín, symbolicky zapsaný $p(malá, černá, benzín)$. V algoritmu se upraví $G$ podle větve pro negativní příklady:




Nyní programu představíme pozitivní příklad velkého, zeleného auta, které jezdí na CNG, symbolicky zapsaný $p(velká, zelená, CNG)$. $G$ bude podle algoritmu upraveno následovně:

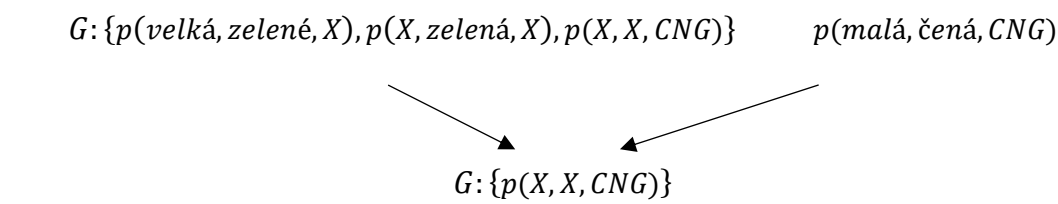



Dále program zpracuje druhý negativní příklad popisující velké, černé auto jedoucí na naftu, symbolicky zapsaný $p(velká, černá, nafta)$, $G$ bude upraveno:




Z důvodu jednoduchosti jsem do $G$ nezapisoval všechny hypotézy, které by vznikly specializací obecnějších hypotéz, jako například hypotézu $p(velká, bílá, X)$ vzniklou specializací z $p(velká, X, X)$.

Nyní program zpracuje poslední pozitivní příklad popisující malé, černé auto jezdící na CNG, symbolicky zapsaný $p(malá, čená, CNG)$. Algoritmus upraví $G$:




V množině $G$ nám po zpracování posledního příkladu zbyla pouze jedna, námi hledaná hypotéza, která popisuje koncept, jako auto, které jezdí na CNG a má jakoukoli barvu a velikost.

### 3.1.5 Hledání od specifického k obecnějšímu

Prohledávání prostoru hypotéz lze provézt i z opačné strany, kdy procházíme prostor od *nejspecifičtější hypotézy* k *hypotéze*, která je nejspecifičtější generalizací dat, tzn. že *hypotéza* pokryje všechny pozitivní instance *konceptu* a žádnou negativní. Algoritmus je specifikován následovně:

Vstup:
Es… množina pozitivních a negativních příkladů konceptu

Výstup:
S…nejspecifičtější obecná hypotéza

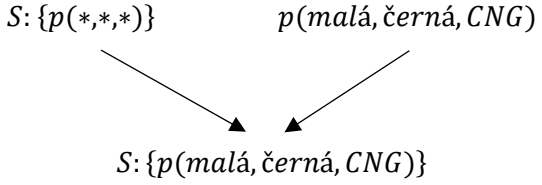
Algoritmus:
BEGIN
Do S přiřaď specifičtější hypotézu konceptu;
N obsahuje všechny použité pozitivní příklady;
FOR EACH negativní příklad n ∈ ES
    BEGIN
    Smaž z S všechny hypotézy, které popisují p;
    Přidej p do N;
    END
FOR EACH pozitivní příklad p ∈ ES
    BEGIN
    FOR EACH s ∈ S, které popisuje n nahraď g nejspecifičtější generalizací, která popisuje n;
    Odstraň z S všechny obecnější hypotézy než jiné hypotézy v G;
    Odstraň z S všechny hypotézy, které popisují nějaký příklad z N;
    END
END

Práci algoritmu si ukážeme na stejném příkladu *konceptu*, jako při opačném průchodu. V tomto příkladu budeme pracovat pouze s pozitivními příklady:

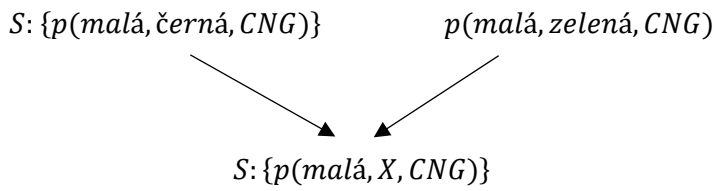
Do *S* nyní vložíme nejspecifičtější hypotézu hledaného konceptu, symbolicky zapsanou $p(*,*,*)$, kde znak $*$ znamená, že žádná hodnota není akceptována atributem. Tato *hypotéza* je natolik specifická, že žádný příklad z prostoru příkladů nebude této hypotéze odpovídat (splňovat), *S* tedy vypadá takto:

$$S: \{p(*,*,*)\}$$

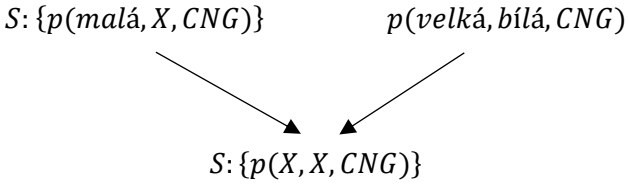
Program zpracuje první pozitivní příklad popisující malé, černé auto, které jezdí na CNG, symbolicky zapsaný $p(malá, černá, CNG)$. Nejspecifičtější hypotéza $p(*,*,*)$ v *S* bude nahrazena sice jinou, proti našemu hledanému konceptu stále příliš specifickou hypotézou, která je však stále obecnější, protože alespoň některé příklady aut této hypotéze budou odpovídat. *S* nyní bude vypadat takto:



Nyní program zpracuje druhý pozitivní příklad popisující malé, zelené auto, které jezdí na CNG, symbolicky zapsané $p(malá, zelená, CNG)$. Představením tohoto příkladu chceme programu říct, že nám nezáleží na barvě auta. Algoritmus upraví hypotézu v *S* následovně:




Posledním pozitivním příkladem popisující velké, bílé auto, které jezdí na CNG, symbolicky zapsané $p(velká, bílá, CNG)$, chceme programu říct, že nám nezáleží ani na velikosti auta. S bude upraveno takto:




Průchodem prostorem hypotéz v opačném směru (od specifického k obecnému) jsme došli ke stejné hypotéze. I když je tato hypotéza konzistentní na příkladech, je to jen jedna z mnoha, které můžou být konzistentní na datech, proto existuje algoritmus, který kombinuje hledání v obou směrech zároveň.

### 3.1.6 Candidate-elimination algoritmus

Candidat-elimination algoritmus, stejně jako průchod prostorem hypotéz od specifického po obecný a naopak, pracuje s *Version space*, kde postupně, pro každý příklad, zužuje tuto množinu *hypotéz konzistentních* na příkladech pomocí specifické a obecné hranice *Version space* zároveň.

Algoritmus je popsaná následovně:

Vstup:
Es… množina pozitivních a negativních příkladů konceptu

Výstup:
S…nejspecifičtější obecná hypotéza

Algoritmus:
BEGIN
Do S přiřaď specifičtější hypotézu konceptu;
Do G přiřaď nejobecnější hypotézu konceptu;
FOR EACH negativní příklad n ∈ ES
    BEGIN
    Smaž z S všechny hypotézy, které popisují p;
    FOR EACH g ∈ G, které popisuje n nahraď g nejobecnější specializací, která nepopisuje n,;
    Odstraň z G všechny specifičtější hypotézy než jiné hypotézy v G;
    Odstraň z G všechny specifičtější hypotézy než jiné hypotézy v S;

```
        END
FOR EACH pozitivní příklad p ∈ ES
        BEGIN
        Smaž z G všechny hypotézy, které popisují p;
        FOR EACH s ∈ S, které popisuje n nahraď g nejspecifičtější generalizací, která popisuje n;
        Odstraň z S všechny obecnější hypotézy než jiné hypotézy v G;
        Odstraň z S všechny obecnější hypotézy než jiné hypotézy v S;
        END
END
```

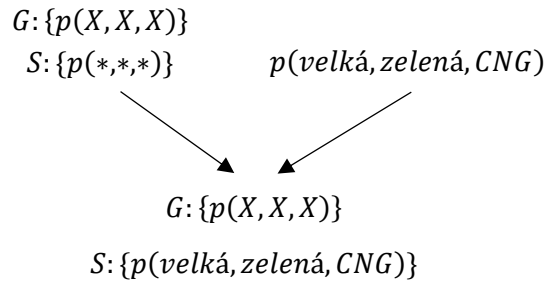
Práci algoritmu si opět ukážeme na učení se konceptu auta. V tomto případě budeme hledat *koncept* auta popisující auto jakékoliv velikosti, které má zelenou barvu a jezdí na CNG.

V prvním kroku algoritmus inicializuje do *G nejobecnější hypotézu* a do *S nejspecifičtější hypotézu*:

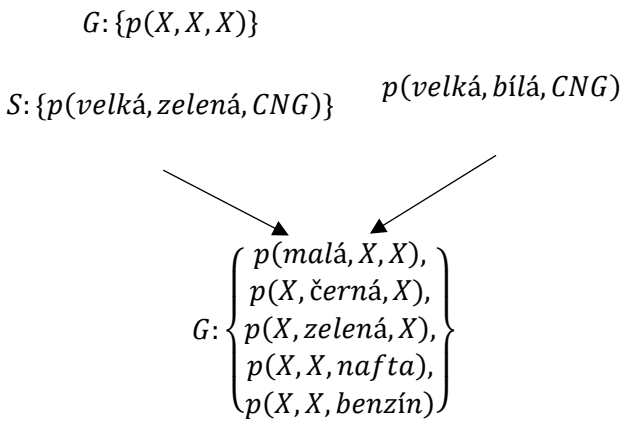
$$G: \{X, X, X\}$$

$$S: \{*,*,*\}$$

Program zpracuje první pozitivní příklad popisující velké zelené auto, jezdící na CNG, *nejspecifičtější hypotéza* v *S* bude nahrazena nejspecifičtější generalizací, které popisuje příklad.




Nyní program zpracuje první negativní příklad velkého bílého auta, které jezdí na CNG. Nejobecnější hypotéza v *G* bude nahrazena nejobecnějšími specializacemi této *hypotézy*, které nepopisují daný negativní příklad:



Programu představíme druhý pozitivní příklad popisující malé zelené auto, jezdící na CNG. Z *G* budou odstraněny všechny *hypotézy*, které nepopisují daný příklad a *hypotéza* v *S* bude nahrazena nejspecifičtější generalizací této *hypotézy*, aby popisovala daný pozitivní příklad:

$$G: \begin{Bmatrix} p(malá, X, X), \\ p(X, černá, X), \\ p(X, zelená, X), \\ p(X, X, nafta), \\ p(X, X, benzín) \end{Bmatrix} \qquad p(malá, zelená, CNG)$$

$$S: \{p(velká, zelená, CNG)\}$$

$$G: \{p(malá, X, X), p(X, zelená, X)\}$$

$$S: \{p(X, zelená, CNG)\}$$

Nyní představíme další negativní příklad. Příklad popisuje velké zelené auto, které jezdí na naftu. Z *G* bude odstraněna *hypotéza* popisující negativní příklad a bude nahrazena jejími nejobecnějšími specializacemi, *hypotéza* $p(malá, X, X)$ bude odstraněna, protože bude obecnější než nově vytvořené *hypotézy*:

$$G: \{p(malá, X, X), p(X, zelená, X)\}$$
$$S: \{p(X, zelená, CNG)\} \qquad p(velká, zelená, nafta)$$

$$G: \{p(X, zelená, CNG), p(X, zelená, benzín)\}$$

$$S: \{p(X, zelená, CNG)\}$$

Posledním negativním příkladem poskytnutým programu, který představuje malé, zelené auto, které jezdí na benzín, odstraníme hypotézu $p(X, zelená, benzín)$, protože popisuje negativní příklad a její specializace by byla specifičtější než hypotéza v *S*:

$$G: \{p(X, zelená, CNG), p(X, zelená, benzín)\}$$
$$S: \{p(X, zelená, CNG)\} \qquad p(malá, zelená, benzín)$$

$$G: \{p(X, zelená, CNG)\}$$

$$S: \{p(X, zelená, CNG)\}$$

Pokud se v *S* a *G* nachází pouze dvě shodné *hypotézy*, znamená to, že se program úspěšně naučil daný *koncept*. Pokud by *S* a *G* po skončení procesu učení byly prázdné, znamenalo by to, že žádná *hypotéza* by nebyla *konzistentní* na příkladech.

Výhodou tohoto algoritmu učení je inkrementální budování *Version space*. To znamená, že pokud se program, který má pro své učení naimplementovaný tento algoritmu, pohybuje v prostředí, které postupně objevuje, může hledat daný koncept "za chodu", budováním *Version space*. Program pracuje s *hypotézami*, které jsou *konzistentní* na do té doby viděných příkladech. Nevýhodou algoritmu je jeho citlivost na šum v datech nebo chybějící data.

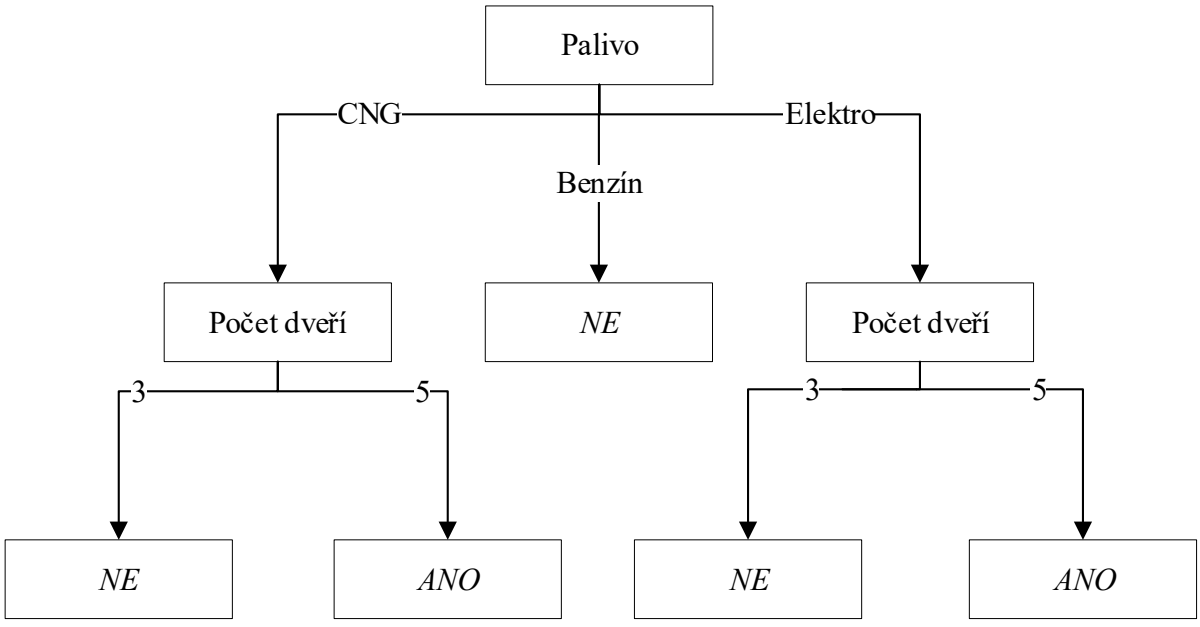
Pro tato problematická data můžeme použít rodinu algoritmů, kterou si představíme v následující kapitole. Jedná se o rodinu algoritmů založenou na rozhodovacích stromech.

## 3.2 Rozhodovací stromy

V [1] uvádí *rozhodovací strom* jako nástroj pro klasifikaci příkladů daného konceptu, ve které hypotéza je reprezentována jako stromová struktura. Jedná se o jednu z nejrozšířenějších metod strojového *učení s učitelem*, která byla použita pro široké spektrum problémů, jako například učení se lékařské diagnózy nebo učení se odhadu rizika při poskytování půjček.

*Rozhodovací strom* se dá chápat jako množinu *pokud-pak* (if-then) pravidel (testů) formovaných do stromové struktury. Stromová struktura začíná v kořeni stromu a postupuje přes jednotlivé uzly k listům. Kořen a každý uzel představuje jeden prvek z množiny if-then testů. Listy stromu představují hodnoty výstupních atributů příkladu. *Rozhodovací stromy* klasifikují příklad postupným testováním hodnot atributů příkladu těmito pravidly, dokud nedojdou k jednomu z listů, jehož hodnota klasifikuje příklad.

Například *rozhodovací strom*, který by představoval *koncept* auta jezdícího na alternativní paliva, které má 5 dveří, má množinu testů obsahující test na palivo a počet dveří, je znázorněn na Obrázku 3.




Obrázek 3: Rozhodovací strom konceptu auta

Každý příklad určený ke klasifikaci by byl podroben sérii testů v pořadí od kořene (test na typ paliva). Pokud by se jednalo o auto jezdící na benzín, příklad by byl rovnou klasifikován, jako negativní instance konceptu. Pokud by auto jezdilo na CNG nebo na elektřinu, bylo by následovně testováno na atribut představující počet dveří, kde by se příklad auta klasifikoval jako instance konceptu, pokud by mělo standardních 5 dveří.

> ***Definice 36:*** *Rozhodovací strom je disjunkce podmínek, kladených na hodnoty vstupních atributů příkladů, které jsou spojeny pomocí konjunkce.*

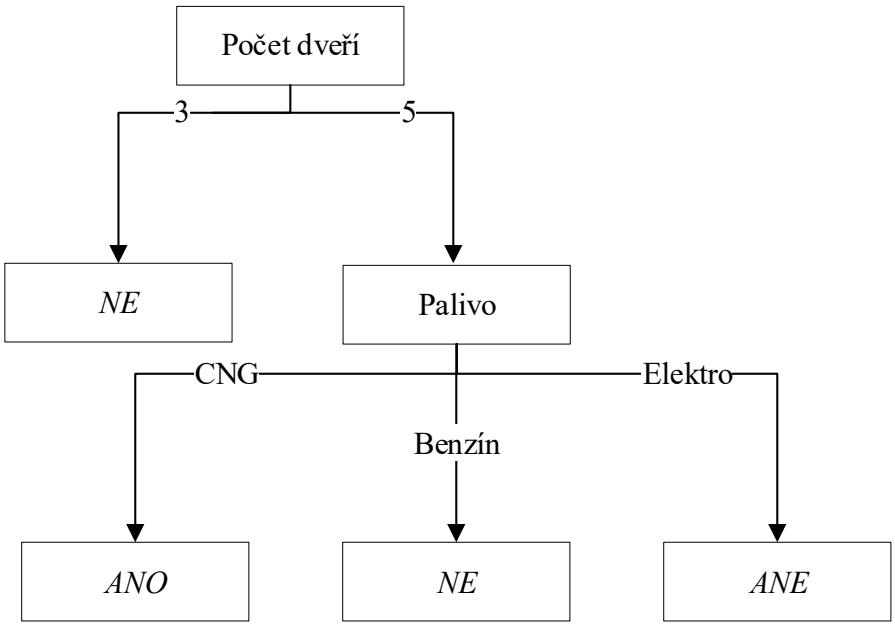
*Rozhodovací strom* na Obrázku 3 by mohl symbolicky zapsaný vypadat takto:

$$\big[[Palivo(x, CNG) \wedge Počet\_dveří(x, 5)] \vee [Palivo(x, Elektro) \wedge Počet\_dveří(x, 5)]\big] \supset Concept(x)$$

Jedním z nejznámějších algoritmů implementující budování rozhodovacích stromů pro klasifikaci příkladů je algoritmus ID3.

### 3.2.1 ID3 algoritmus

Jak bylo zmíněno výše, *rozhodovací strom* se skládá z množiny testů na hodnoty atributů v příkladu. Každý příklad je podroben sérii testů, které skončí u některého z listů v některé větvi daného stromu. Tato cesta se však může lišit v pořadí testů, které se provádí, mezi různými stromy, které však popisují stejný koncept. Změnou pořadí testů získáme různé stromy lišící se velikostí. Změnou pořadí testů na hodnoty atributu z rozhodovacího stromu na Obrázku 3 můžeme získat rozhodovací strom znázorněný na Obrázku 4.




Obrázek 4: Varianta rozhodovacího stromu z Obrázku 3

*Rozhodovací strom* na Obrázku 4 bude klasifikovat příklady stejně správně, jako rozhodovací strom na Obrázku 3. Tento strom je v

Má však méně listů, na rozdíl od stromu na Obrázku 3, protože má jen 4 listy. Při budování *rozhodovacího stromu* tedy musíme uvažovat nad tím, který z různě velikých stromů popisují *koncept*, bude nejlépe klasifikovat i neviděné příklady. Jinými slovy, jak vybrat posloupnost testů, abychom získal nejlepší *hypotézu* (rozhodovací strom), které bude *konzistentní* i na neviděných příkladech.

Tuto problematiku řeší algoritmus ID3 tak, že nejvhodnější strom je ten, který je nejjednodušší. Abychom dostali nejjednodušší strom, musíme testy řadit podle určitého kritéria, kterým je, jak dobře daný test rozdělí data. Takovému kritériu se říká *informační zisk*.

Informační zisk se definuje pomocí *entropie*.

> ***Definice 37****: [3] Entropie je míra nejistoty náhodné veličiny (v našem případě míra nesourodosti příkladů podle hodnoty výstupního atributu příkladu). Entropie se udává v bitech.*

*Entropie* v množině příkladů, kde jsou všechny stejného typu (náhodná veličina v našem případě je, zda se jedná o pozitivní nebo negativní příklad), je nula bitů, protože zde není žádná nejistota (různorodost), tzn. pozorováním této hodnoty nezískáme žádnou informaci. Pokud máme množinu, ve které je stejný poměr pozitivních a negativních příkladů, *entropie* této množiny by byl 1 bit, tzn. potřebovali bychom 1 bit pro popis negativních a pozitivních příkladů ve dvojkové soustavě.

> ***Definice 38:*** *[1] Entropie je obecně definována jako:*
>
> $$E(P) = \sum_{i=1}^{c} -p_i \, log_2 \, p_i$$
>
> *kde $E(P)$ je entropie množiny $P$, $p_i$ je poměr počtu prvků v množině $P$ ku třídě $c$. Třída $c$ v našem konceptuálním učení představuje výstupní atribut konceptu nabývající hodnot True nebo False.*

*Informační zisk* nám udává pokles *entropie* v podmnožinách vzniklými rozřazením příkladů pomocí daného testu hodnoty atributu.

> ***Definice 39:*** *[1] Informační zisk $Gain(S, A)$, kde $S$ je množina příkladů a $A$ je atribut testu, u kterého se zjišťuje informační zisk, je dán vzorcem:*
>
> $$Gain(S, A) = E(S) - \sum_{i=1}^{n} \frac{|c_i|}{|c|} E(c_i)$$
>
> *Kde $|c|$ je počet všech možných hodnot atributu $A$ v dané množině příkladů $S$, $|c_i|$ je počet příkladů, u kterých má v množině $S$ atribut $A$ hodnotu $c_i$ a $E(c_i)$ je entropie v příkladech, které mají hodnotu $c_i$ atributu $A$.*

Mějme množinu *P* pozitivních a negativních instancí konceptu auta, které má 5 dveří a jezdí na alternativní palivo(CNG, Elektřina) popsaných v Tabulce 1:

| ID | Počet dveří | Typ paliva | Instance konceptu |
|---|---|---|---|
| **1** | 3 | CNG | *False* |
| **2** | 5 | CNG | *True* |
| **3** | 3 | Benzín | *False* |
| **4** | 3 | Elektro | *False* |
| **5** | 5 | Elektro | *True* |

Tabulka 1: Množina S pozitivních a negativních příkladů konceptu auta

*Entropie* množiny *S* je tedy:

$$E(S) = -\frac{2}{5}\, log_2\left(\frac{2}{5}\right) - \frac{3}{5}\, log_2\left(\frac{3}{5}\right)$$

$$E(S) = 0{,}97\ bit\text{ů}$$

Když máme míru různorodosti na množině příkladů, můžeme přistoupit k určování, který test atributu je nejvhodnější pro klasifikování dat pomocí *informačního zisku*.

Budeme chtít rozhodnout, zda v našem příkladě bude jako první test (test v kořeni rozhodovacího stromu) na počet dveří auta nebo na typ paliva. Podle Tabulka 1 si vypočítáme pro oba atributy *informační zisk*. Začněme nejdříve informačním ziskem atributu typu paliva:

$$Gain(S, Typ\_paliva) = E(S) - \left(\frac{2}{5} * E(CNG) + \frac{2}{5} * E(Elektro) + \frac{1}{5} * E(Benzín)\right)$$

$$E(CNG) = -\frac{1}{2} * \log_2 \frac{1}{2} - \frac{1}{2} * \log_2 \frac{1}{2}$$

$$E(CNG) = 1\ bit$$

$$E(Elektro) = -\frac{1}{2} \log_2 * \frac{1}{2} - \frac{1}{2} * \log_2 \frac{1}{2}$$

$$E(Elektro) = 1\ bit$$

$$E(Benzín) = -1 * \log_2 1 - 1 * \log_2 1$$

$$E(CNG) = 0\ bit\text{ů}$$

$$Gain(S, Typ\_paliva) = 0{,}97 - \left(\frac{2}{5} * 1 + \frac{2}{5} * 1 + \frac{1}{5} * 0\right)$$

$$Gain(S, Typ\_paliva) = 0{,}17\ bit\text{ů}$$

Nyjí vypočítáme *informační zisk* pro atribut Počet dveří:

$$Gain(S, Počet\_dveří) = E(S) - \left(\frac{3}{5} * E(3\ dveře) + \frac{2}{5} * E(5\ dveří)\right)$$

$$E(3\ dveře) = -\frac{0}{5} * \log_2 \frac{0}{5} - \frac{3}{5} * \log_2 \frac{3}{5}$$

$$E(3\ dveře) = 0{,}442\ bitu$$

$$E(5\ dveří) = -\frac{2}{5} * \log_2 \frac{2}{5} - \frac{0}{5} * \log_2 \frac{0}{5}$$

$$E(5\ dveří) = 0{,}529\ bitu$$

$$Gain(S, Počet\_dveří) = 0{,}97 - \left(\frac{3}{5} * 0{,}442 + \frac{2}{5} * 0{,}259\right)$$

$$Gain(S, Počet\_dveří) = 0{,}493\ bitu$$

Z výpočtů vyplývá, že při vytváření *rozhodovacího stromu* by nejefektivnější volbou testu v kořeni stromu byl test hodnoty atributu *Počet_dveří*, protože jeho použitím získáme větší *informační zisk* 0,493, oproti *informačnímu zisku* z testu na typ paliva 0,17. *Rozhodovací strom* na Obrázku 4 je tedy vhodnější pro klasifikování i neviděných obrázku, než rozhodovací strom znázorněný na Obrázku 3.

Algoritmus ID3 je podle [4] specifikován následovně:

```
Vstup:
Es… množina pozitivních a negativních příkladů konceptu
As…množina všech atributů, nacházejících se v příkladech

Výstup:
T…rozhodovací strom

Algoritmus:
BEGIN
    IF všechny příklady v Es patří do stejné třídy c (mají stejnou hodnotu výstupního atributu) THEN
        v T vytvoř kořen stromu označený c;
    ELSE IF As je prázdný THEN
        v T vytvoř kořen stromu označený disjunkcemi všech příkladů v Es;
    ELSE
        vyber atribut a z As a udělej z něj kořen stromu T;
        FOR EACH hodnotu v atributu a
            vytvoř větev označenou V;
            V = větev stromu vytvořena rekurzivním spuštěním algoritmu na podmnožinu příkladů
            obsahující atribut a s hodnotou v;
        END
    END IF
END
```

Při každém výběru atributu provádí podle [1] algoritmus ID3 přesné vyhodnocování, který atribut je aktuálně nejvhodnější pro testování a induktivně buduje *hypotézu*. Stejně jako u každého induktivního učení i *rozhodovací stromy* pracují v nějakém prostoru hypotéz, ve kterém se při učení vytváří podmnožina *hypotéz*, které jsou *konzistentní* na datech, z předchozích kapitol známá jako V*ersion space*.

V takové prostoru hypotéz se provádí hledání hypotézy (rozhodovacího stromu), která je konzistentních na datech poskytnutých učitelem, od prázdného stromu, k finálnímu.

Oproti Candidate-elimination algoritmu, který si udržoval množinu všech *konzistentních hypotéz*, však ID3 udržuje pouze jednu *konzistentní hypotézu*. ID3 má však oproti Candidate-elimination algoritmu výhodu v tom, že dokáže pracovat s daty obsahující šum. Pokud algoritmus během učení narazí na příklad, kterému chybí hodnota některého vstupního algoritmu, může nahradit chybějící hodnotu atributu nejčastěji se vyskytující hodnotou atributu mezi všemi příklady poskytnutými učitelem nebo může použít komplexnější metodu využívající pravděpodobnost výskytu hodnoty daného atributu.

ID3 algoritmus se však potýká s problematikou zvanou *overfitting.*

> ***Definice 40:*** *[2] Overfitting je jev, který nastane, když agent vytvoří hypotézu, která byla vybudována na základě pravidelnosti vyskytující se pouze v příkladech poskytnutých učitelem, ale nevyskytující se v neviděných příkladech.*

Jedním z řešení této problematiky může být takzvané *ořezávání stromu* (tree pruning), konkrétně algoritmus *rule post-pruning*, který strom, vzniklý učením, rozebere na jednotlivé cesty od kořene k listu a tyto cesty (větve stromu) upravuje za účelem zpřesnění efektivnosti následné klasifikace a odstranění overfitingu. Více o ořezávání stromů se dočtete v [17], o rozhodovacích stromech samotných se více dočtete v [1].

## 3.3 Metody strojového učení bez učitele

Doposud byly v předchozích kapitolách popsány metody strojového učení, kde existovala nějaká forma učitele, ať už to byl člověk nebo nějaký program, který programu poskytoval klasifikované příklady, tj. příklady s výstupními atributy a jejich hodnotami. U metod strojového *učení bez učitele* je na programu samotném, aby klasifikoval příklady. Strojové *učení s učitelem* by se dalo přirovnat k žákům ve škole a jejich učiteli zeměpisu (žáci se o nějakém kontinentu učí od učitele), kdežto strojové *učení bez učitele* by se dalo přirovnat k objeviteli, který prozkoumává neznámý kontinent (program se o kontinentu musí naučit sám).

Jak již bylo zmíněno v Kapitole 1, problematika strojového *učení bez učitele* spočívá v rozdělení příkladů do shluků (clusterů), které predikují hodnoty atributů pro každý příklad daného shluku. Cílem je rozdělit příklady tak, aby splňovaly nějaký kvalitativní standard dané třídy, například nějakou míru podobnosti mezi příklady.

### 2.3.1 Numerická taxonomie

Jednou z nejstarších metod pro strojové učení bez učitele je takzvaná *numerická taxonomie*, známá také jako aglomerativní shlukování.

> ***Definice 41****: Algomerativní shlukování je metoda hierarchické dekompozice dat založená na podobnosti mezi daty.*

Jedná se o metodu založenou na příkladech, které jsou popsané pomocí kolekce atributů s numerickými hodnotami, které představují n-dimenzionální vektor atributů v n-dimenzionálním vektorovém prostoru. Tyto příklady jsou následně hodnoceny mezi sebou kritériem podobnosti v podobě euklidovské vzdálenosti v daném n-dimenzionálním prostoru. Na základě této podobnosti se všechny příklady

shlukují a výsledné shluky definují příklady do nich spadající. Výsledkem této metody je binární strom, ve kterém listy stromu jsou instance a uzly představují shluky.

Tuto metodu lze uplatnit i na symbolickou reprezentaci. U symbolické reprezentace však musíme použít jiný typ určování podobnosti mezi příklady. Takový typ může být například založen na poměru počtu shodných hodnot symbolicky zapsaných atributů mezi příklady.

Jako příklad opět použijeme *koncept* aut, kde auto je popsané pomocí vstupních atributů typu paliva, počtu dveří, stupně výbavy a barvy auta. Jednotlivé příklady jsou opět symbolicky zapsány jako množina hodnot, které u daného příkladu nabývají vstupní atributy. Mějme tyto příklady:

$$auto1: \{CNG, 3, střední, bílá\}$$

$$auto2; \{CNG, 5, střední, modrá\}$$

$$auto3: \{benzín, 5, základní, červená\}$$

$$auto4: \{benzín, 5, nejvyšší, červená\}$$

Výše zmíněná metrika by mohla tyto příklady vyhodnotit takto:

- Podobnost mezi $auto1$ a $auto2$ je $\frac{2}{4}$
- Podobnost mezi $auto1$ a $auto3$ je 0
- Podobnost mezi $auto1$ a $auto2$ je rozdílná od podobnosti mezi $auto3$ a $auto4$

U definování *konceptů*, do kterých by příklady spadaly, však tato metrika nestačí, protože při porovnávání příkladů mají všechny atributy stejnou váhu. Když člověk přirozeně shlukuje objekty podle podobnosti, často intuitivně přiřazuje některým atributům větší váhu než jiným, a tím docílí jiného rozdělení příkladů do shluků, než by docílil agent s metrikou bez vah mezi atributy. Tyto aglomerativní shluky neposkytují žádný sémantický popis vytvořených shluků. Shluky jsou popsány extenzionálně, což znamená popis shluku výčtem jeho prvků.

Tento problém řeší, podle [4] metoda *konceptuálního shlukování*, která vytváří nad shluky jejich obecnou definici (popis konceptu). Zástupcem této metody je algoritmus CLUSTER/2, který používá pro vytváření obecných popisů konceptů operace pro manipulaci jazyka reprezentace. CLUSTER/2 vytváří nastavitelný $k$ počet shluků vytvořených kolem takzvaných jader (vybraných příkladů příkladů). Algoritmus poté vyhodnocuje podle kvalitativních kritérií shluky a případně proces opakuje, dokud nesplní kvalitativní kritéria. Algoritmus je popsán takto:

1. Algoritmus zvolí $k$ jader náhodně nebo podle stanovené funkce.
2. Pro každé jádro stanoví dané jádro jako pozitivní příklad a ostatní jádra jako negativní příklad konceptu. Pro dané jádro vytvoří hypotézu tak obecnou, aby popisovala všechny pozitivní příklady.

3. Algoritmus dále klasifikuje všechny ostatní dosud neklasifikované příklady podle hypotéz vytvořených z jader (shluky se mohou překrývat).
4. Hypotézu každého shluku následně algoritmus upraví tak, aby byla specifická natolik, že popisuje jen příklady spadající do daného shluku (shluky se nepřekrývají).
5. Algoritmus následně podle nějaké metriky (například výše zmíněná podobnost) příklad, který se nachází nejvíce uprostřed každého shluku a vybere ho za nové jádro.
6. Algoritmus pro nové jádra opakuje 2. až 5. krok, dokud hypotézy shluků nesplňují nějaké kvalitativní kritérium, například složitost (jednoduchost) jednotlivých hypotéz.

Pokud se u vytváření shluků po několikáté iteraci neprojevuje zlepšení, je například možné změnit výběr jader v kroku 5 na výběr nového jádra na kraji shluku.

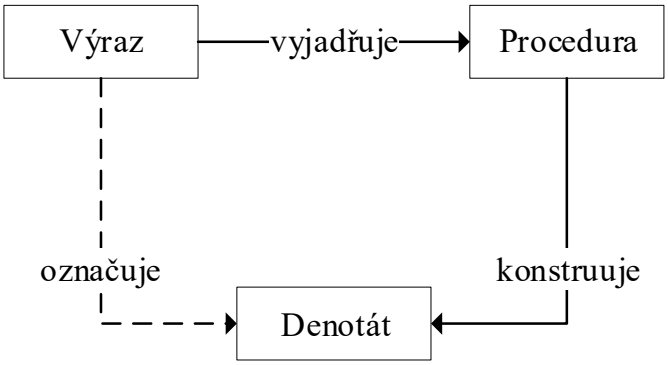
O konceptuálním shlukování se více dočtete v [18] a [19].

# 4 Transparentní intenzionální logika

V této kapitole je popsána Transparentní intenzionální logika (TIL), která byla vytvořena českým logikem Pavlem Tichým, kterou formuloval ve své knize [20].

V [21] se TIL uvádí jako typovaný λ-kalkul, používající procedurální sémantiku.

> ***Definice 42:*** *Procedurální sémantika přiřazuje významům abstraktní algoritmicky strukturované procedury, představující jejich význam. Takovéto procedury konstruují denotát (objekt, o kterém výraz mluví), označovaný daným výrazem. Viz obr.5.*




Obrázek 5: Sémantické schéma procedurální sémantiky TIL

Zmíněné abstraktní algoritmicky strukturované procedury jsou v TIL definované takzvanými *konstrukcemi*. *Konstrukce* konstruují denotát, jehož základním typem je funkce.

> ***Definice 43:*** *Funkce je zobrazení z množiny D(definiční obor), do množiny H(obor hodnot), které přiřazuje každému prvku z D nanejvýš jeden prvek z H.*

Tyto funkce jsou však parciální[10], protože existují výrazy, které mají význam, ale nemají denotát. Například výraz : "2 : 0". Výhodou tohoto funkcionálního přístupu je, že na funkcích můžeme provádět operace, např. operaci *aplikace* funkce na argument nebo operaci *abstrakce* od hodnot argumentů funkce.

Denotátem výrazu může být entita jakéhokoliv typu (i *konstrukce*). Pro usnadnění hledání adekvátní *konstrukce*, která by vyjadřovala daný výraz, jsou entity ontologicky rozřazeny. Tato ontologie je uspořádána do tzv. *rozvinuté hierarchie typů*.

Nejjednodušší typem entit v *rozvinuté hierarchii typů* jsou *typy řádu 1* (entity, které nejsou konstrukce). *Typy řádu 1* jsou v [21] definovány následovně:

> ***Definice 44:*** *Nechť B je báze, což znamená kolekce vzájemně disjunktních neprázdných množin. Pak:*
>
> *i. Každý prvek b je atomický (elementární) typ řádu 1 nad B.*

[10] Parciální funkce je funkce, která pro některé prvky v definičním oboru nemá zobrazeni v oboru hodnot.

ii. *Nechť $\alpha, \beta_1, \ldots, \beta_m$ ($m > 0$) jsou typy řádu 1 nad B. pak kolekce $(\alpha\beta_1, \ldots, \beta_m)$ všech m-árních parciálních funkcí, tj. zobrazení z kartézského součinu , $\beta_1 \times \ldots \times \beta_m$ do $\alpha$, je molekulární (neboli funkcionální) typ řádu 1 nad B.*

iii. *Nic jiného není typem řádu 1 nad B než dle (i) a (ii).*

V [21] je uvedeno, že nejvhodnější objektová báze, použitá při zpracování přirozeného jazyka, se osvědčila báze obsahující:

- *Pravdivostní hodnoty o*: Jedná se o dvouprvkovou množinu {P, N}, kde P přiřazujeme pojmu *Pravda* a N pojmu *Nepravda*.
- *Časové okamžiky/ Reálná čísla τ*: Jedná se množinu obsahující nekonečně mnoho prvků odpovídající časovým okamžikům, které lze chápat i jako reálná čísla.
- *Možné světy ω*: Jedná se o množinu logicky možných světů.
- *Universum ι*: Jedná se o množinu individuí.

Podle [22] jsou dalším typem entit v *rozvětvené hierarchii typů* již výše zmíněné *konstrukce*. *Konstrukce* operují nad entitami určitého typu za účelem vytvoření objektu typu určeného konstrukcí. Existují dva typy *konstrukcí*, atomické a molekulární.

V TIL jsou definovány dvě atomické konstrukce: *trivializace* a *proměnná*. Funkcionalita *trivializace* je podobná funkcionalitě pointeru v programovacích jazycích, což je pouze ukazatel objekt. *Trivializace* objektu A, značeno ${}^0$A, pouze konstruuje objekt A. *Proměnná* je *konstrukce*, která konstruuje objekty vhledem k jejich valuaci (*proměnné v-konstruují* objekty, kde *v* je parametr valuace).

*Molekulární konstrukce* jsou *konstrukce*, které mohou obsahovat i jiné konstituenty[11] (podkonstrukce). Mezi molekulární *konstrukce* v TIL patří *Kompozice*, *Uzávěr*, *Provedení* a *Dvojí Provedení*.

*Konstrukce* jsou v [21] definovány následovně:

***Definice 45:***

i. *Proměnná x je konstrukce, která konstruuje objekt O příslušného typu v závislosti na valuaci v; tedy x v-konstruuje O*

ii. *Trivializace: Je-li X jakýkoliv objekt(extenze[12], intenze[13] nebo i konstrukce), ${}^0X$ je konstrukce zvaná Trivializace. Konstruuje objekt X bez jakékoliv změny.*

iii. *Kompozice $[X\ Y_1 \ldots Y_m]$ je konstrukce: Je-li X konstrukce, která v-konstruuje funkci f typu $(\alpha\beta_1, \ldots, \beta_m)$ a $Y_1, \ldots, Y_m$ v-konstruují po řadě objekty $B_1, \ldots, B_m$ typů $\beta_1, \ldots, \beta_m$, pak Kompozice $[X\ Y_1 \ldots Y_m]$ v-konstruuje hodnotu funkce f na argumentech $B_1, \ldots, B_m$ (tj. objekt typu $\alpha$, pokud f má $<B_1, \ldots, B_m>$ hodnotu). Jinak je Kompozice $[X\ Y_1 \ldots Y_m]$ v-nevlastní, tj. ne (v-)konstruuje žádný objekt.*

[11] Objekt, na kterém konstrukce operuje může být i jiná konstrukce. Takovým objektům říkáme konstituenty.
[12] (α-)extenze jsou prvky typu α, kde $\alpha \neq (\beta\omega)$ pro libovolný typ β; tedy extenze jsou α-objekty jejichž doménou není množina možných světů.
[13] (α-)intense jsou prvky typu $(\alpha\omega)$, tedy funkce z možných světů do libovolného typu α.

iv. *Uzávěr* $[\lambda x_1 \ldots x_m\ Y]$ *je konstrukce. Nechť* $x_1, x_2 \ldots x_m$ *jsou navzájem různé proměnné, které v-konstruují po řadě objekty typu* $\beta_1, \ldots, \beta_m$*, a nechť Y je konstrukce, která v-konstruuje α-objekt. Pak* $[\lambda x_1 \ldots x_m\ Y]$ *v-konstruuje funkci* $f/(\alpha\beta_1, \ldots, \beta_m)$ *a to takto: Nechť* $v(B_1/x_1, \ldots, B_m/x_m)$ *je valuace, která se liší od valuace v nanejvýš tím, že přiřazuje objekty* $B_1/\beta_1, \ldots, B_m/\beta_m$ *proměnným* $x_1 \ldots x_m$*. je-li Y* $v(B_1/x_1, \ldots, B_m/x_m)$*-nevlastní (viz iii), pak funkce f není definována na* $<B_1, \ldots, B_m>$*. Jinak je hodnotou funkce f argumentu* $<B_1, \ldots, B_m>$ *α-objekt* $v(B_1/x_1, \ldots, B_m/x_m)$*-konstruovaný Y.*

v. *Provedení* $^1X$ *je konstrukce, která buď v-konstruuje objekt v-konstruovaný konstrukcí X, nebo pokud X není konstrukce nebo je v-nevlastní, je rovněž* $^1X$ *v-nevlastní, tj. nekonstruuje žádný objekt.*

vi. *Dvojí provedení* $^2X$ *je konstrukce. Tato konstrukce je v-nevlastní, pokud X není konstrukce, nebo pokud X ne v-konstruuje jinou konstrukci, nebo v-konstruuje v-nevlastní konstrukci. Jinak jestliže X v-konstruuje konstrukci Y a Y v-konstruuje objekt Z, pak* $^2X$ *v-konstruuje Z.*

vii. *Nic jiného není konstrukce než dle (i)-(vi).*

*Konstrukce* nemůžou být stejného typu jako objekt, který konstruují. Podle Definice 41 entity typu řádu 1 nejsou *konstrukce*, proto *konstrukce* konstruující entity typu řádu 1 musí být alespoň typu řád 2. Pokud nějaká *konstrukce* konstruuje entitu řádu 2, musí být tato *konstrukce* být alespoň typu řád 3 atd. *Rozvětvená hierarchie typů řádů nad bází B* je proto podle [21] induktivně definována jako:

***Definice 46:***

$T_1$ *(typy řádu 1) jsou definovány v Definici 41.*

$C_n$ *(konstrukce řádu n)*

i. *Nechť x je proměnná, která v-konstruuje objekty typu řádu n. Pak x je konstrukce řádu n nad B.*

ii. *Nechť X je prvek typu řádu n. pak* $^0X$, $^1X$, $^2X$ *jsou konstrukce řádu n nad B.*

iii. *Nechť* $X, X_1, \ldots, X_m$ $(m > 0)$ *jsou konstrukce řádu n nad B. Pak* $[X\ X_1, \ldots, X_m]$ *je konstrukce řádu n nad B.*

iv. *Nechť* $x_1, x_2 \ldots x_m, X$ $(m > 0)$ *) jsou konstrukce řádu n nad B. Pak* $[\lambda x_1 \ldots x_m\ X]$ *je konstrukce řádu n nad B.*

v. *Nic jiného není konstrukce řádu n nad B než to, co je definováno dle* $C_n$ *(i)-(iv).*

Při analýze výrazů v TIL narazíme na dva druhy výrazů: *empirické* a *analytické*. *Empirické* výrazy jsou podle [21] definovány takto:

***Definice 47:*** *Výraz, který označuje nekonstantní intenzi, tj. intenzi, která alespoň ve dvou „světamizích“* $<w, t>$ *nabývá různých hodnot, nazveme empirickým výrazem.*

*Analytické* výrazy jsou podle [21] definovány následovně:

***Definice 48:*** *Analytický výraz je výraz, jehož denotátem je buďto extenze nebo konstantní intenze nebo nemá v žádném <w, t> denotát.*

Příkladem *analytických* výrazů jsou například matematické objekty řád 1, jako množina individuí (οι), binární funkce sčítání, odčítání, násobení, dělení (τττ) atd.

Logické spojky ∧ (konjunkce), ∨ (disjunkce), ⊃ (implikace) jsou typu (ooo), logická negace ¬ je typu (oo).

Nejčastějšími *empirickými* výrazy typu řádu 1 (intenzemi typu řádu 1), které neobsahují *konstrukce* jsou:

- *Individuové úřady*(role): Jedná se o objekty typu ((ιτ)ω)[14], zkráceně psáno $\iota_{\tau\omega}$, například „nejchytřejší student“, „policejní prezident“, „rektor VŠB“ atd.
- *Vlastnosti individuí*: Jedná se o objekty (((οι)τ)ω), zkráceně psáno $(o\iota)_{\tau\omega}$, například výraz „vysoký“, „bohatý“, „ženatý“ atd.
- *Empirické funkce* či *atributy*: Jedná se o objekty typu (((αβ)τ)ω), zkráceně psáno $\alpha\beta_{\tau\omega}$, například výraz „žalobce (někoho)“ typu $(\iota)_{\tau\omega}$, „osobní strážce (někoho)“ „řidič (něčeho)“ atd.
- *Propozice*: Jedná se o věty typu (((ο)τ)ω), zkráceně psáno $o_{\tau\omega}$, například „Slunce svítí.“

Pro analýzu přirozeného jazyka jsou *kvantifikátory* nezbytné. V TIL , podobně jako v predikátové logice prvního řádu, existují všeobecný kvantifikátor $\forall^{\alpha}$ a existenční kvantifikátor $\exists^{\alpha}$. Oba jsou typu (o(oα)), kde α je libovolný typ. *Kvantifikátory* jsou dle [21] definovány následovně:

***Definice 49:*** *(Neomezené[15]) kvantifikátory $\forall^{\alpha}$, $\exists^{\alpha}$ jsou typově polymorfní funkce typů (o(oα)) definované takto:*

*i. Všeobecný kvantifikátor $\forall^{\alpha}$ je funkce, která přiřazuje třídě α-prvků C/(oα) hodnotu P, jestliže C obsahuje všechny prvky typu α, jinak N.*

*ii. Existenční kvantifikátor $\exists^{\alpha}$ je funkce, která přiřazuje třídě α-prvků C/(oα) hodnotu P, jestliže C je neprázdná, jinak N.*

Jako příklad analýzy přirozeného jazyka použijeme větu „Spider-man je hrdina New Yorku.“. Nejdříve provedeme tzv. typovou analýzu, která jednotlivým objektům zmíněných ve větě:

Typová analýza:

$NewYork\ /\iota$

$Hrdina/\left((o\iota)\iota\right)_{\tau\omega}$

$Spiderman/\iota_{\tau\omega}$

[14] Roli nebo úřad mohou zastávat různá individua v různých možných světech (časový sled množin různých vzájemně si neodporujících empirických faktů), proto je daný výraz “uzamknut” v daném světě pomocí *modálního operátoru* ω. Individua danou roli nebo úřad mohou zastupovat různě dlouhý časový okamžik, proto je výrazu “uzamknut“ v daném časovém okamžiku pomocí *temporálního parametru* τ.

[15] Omezené kvantifikátory jsou polymorfní funkce typu (o(oα) (oα)) podrobněji popsané v [21].

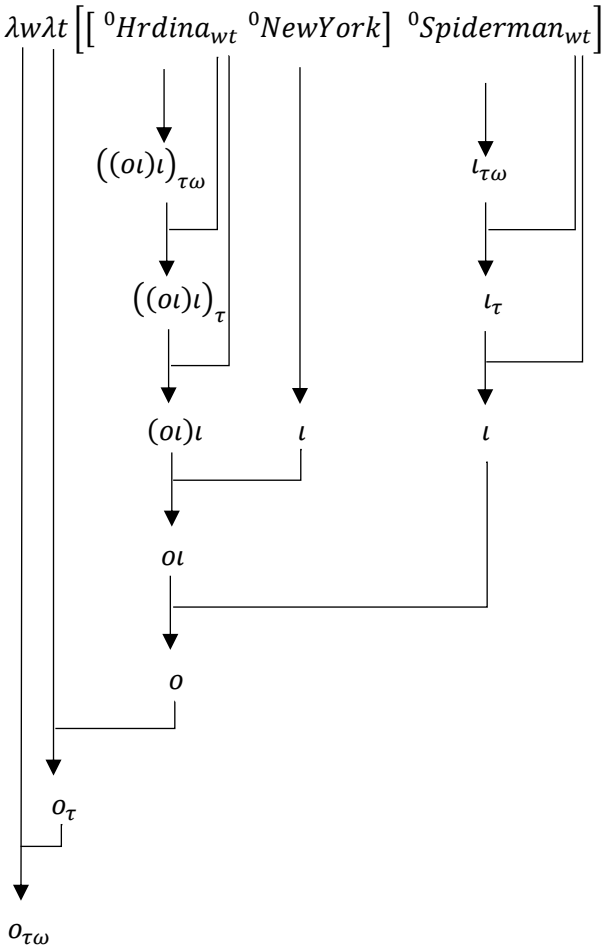
$Celá\ věta\ je\ propozice\ /o_{\tau\omega}$

*Konstrukce* $Hrdina\ (nečeho)$ musí být nejdříve extenzionalizována, tj. aplikovaná na daný svět *w* a na časový okamžik *t* až poté může být *konstrukce* vyhodnocena. Jelikož je celá věta typu $o_{\tau\omega}$, je nutné na závěr abstrahovat od hodnot *w* a *t*.

Nyní můžeme provézt syntézu věty:

$$\lambda w \lambda t \left[ \left[ {}^0Hrdina_{wt}\ {}^0NewYork \right]\ {}^0Spiderman_{wt} \right]$$

Po každé syntéze je vhodné provézt typovou kontrolu, zda jsme provedli syntézu v souladu s typovými pravidly:




Transparentní intenzionální logika, jakožto nástroj pro analýzu přirozeného jazyka, je dále podrobně popsaná v publikacích [23], [21], další informace o TIL naleznete na webových stránkách kurzu zabývajícího se zpracováním přirozeného jazyka prof. Duží z VŠB-TU v Ostravě [24].

# 5 Syntax jazyka TIL-Script

V této kapitole je popsán jazyk TIL-Script jako nástroj nahrazující TIL, jejíž notace není vhodná pro zpracování počítačem.

> ***Definice 50:*** *„TIL-Script je funkcionální deklarativní jazyk operačně izomorfní s TIL.“ [25, s. 9]*

V [26] uvádějí, že jazyk TIL je pro zpracování počítačem nevhodný z několika důvodů:

- Notace konstrukcí v TIL není plně standardizována.
- Abeceda TIL je obsáhlejší o speciální znaky, které nejsou obsaženy ve standardu ACII.
- TIL neurčuje rozhraní pro ontologie, jazyk obsahu je limitován použitím konceptů dané domény.
- Neexistuje standardizovaná typová báze, [21] volba a obsah báze závisí na oblasti, která je předmětem zkoumání.

*TIL-script* pokrývá veškerou funkcionalitu TIL s modifikací zápisu tak, aby veškerá notace byla zapsaná pomocí standardu ACII. Typová báze je navíc obohacena o typy, které se používají ve funkcionálním programování.

Typová báze v *TIL-Script* jazyku je definována, podle [25], Tabulkou 2.

| **TIL** | **Popis** | **TIL-Script** |
|---|---|---|
| o | Pravdivostní hodnoty | Bool |
| ι | Individua (universum) | Indiv |
| τ | Časové okamžiky | Time |
| ω | Možné světy | World |
| | Celá čísla | Int |
| τ | Reálná čísla | Real |
| | Řetězec znaků | String |
| α | Nespecifikovaný typ | Any |
| **n* | Konstrukce řádu n | * |

Tabulka 2: Typová báze jazyka TIL-Script

Zápis konstrukcí v jazyce *TIL-Script* je, podle [26], definován v Tabulce 3.

| **TIL** | **Popis** | **TIL-Script** |
|---|---|---|
| $^0X$ | Trivializace | 'C |
| x | Proměnná | X nebo x:Typ |
| $\lambda x_1 \ldots x_m\ Y$ | Uzávěr | \x1…\xm C |
| $[X\ Y_1 \ldots Y_m]$ | Kompozice | [C C1 … Cm] |
| $C_{wt}$ | Intenzionální sestup | C@w,t |
| $^1C$ | Provedení | ^1C |
| $^2C$ | Dvojí Provedení | ^2C |

Tabulka 3: Zápis konstrukcí v jazyce TIL-Script

Celá gramatika jazyka *TIL-Script* je v [25] dána následovně:

- Start = {definice typu, ukončení | definice entity, ukončení | konstrukce, ukončení | definice objektu, ukončení | definice globální proměnné, ukončení};
- ukončení = volitelný bílý znak, ".", volitelný bílý znak;
- definice typu = "TypeDef", bílý znak, název typu, volitelný bílý znak, ":=", volitelný bílý znak, datový typ;
- definice objektu = "ObjectDef", bílý znak, název objektu, volitelný bílý znak, ":=", volitelný bílý znak, konstrukce;
- definice entity = název entity, { volitelný bílý znak,",", volitelný bílý znak, název entity}, volitelný bílý znak, "/", volitelný bílý znak, (datový typ | název typu);
- konstrukce = trivializace | proměnná | kompozice | uzávěr | provedení | dvojí provedení;
- definice globální proměnné = název proměnné, { volitelný bílý znak, ",", volitelný bílý znak, název proměnné}, volitelný bílý znak, "->", volitelný bílý znak, datový typ;
- datový typ = "Bool" | "Indiv" | "Time" | "String" | "World" | "Real" | "Int" | "Any" | "List", volitelný bílý znak, "(", volitelný bílý znak, datový typ, volitelný bílý znak, ")" | "Tuple", volitelný bílý znak, "(", volitelný bílý znak, datový typ, volitelný bílý znak, ")" | "*" | "(", volitelný bílý znak, datový typ1, volitelný bílý znak ")" | "(", datový typ1, bílý znak, datový typ1,")" | "(", datový typ, datový typ, ")" | datový typ,'@tw';
- datový typ1 = datový typ1, bílý znak, datový typ1 | datový typ;
- proměnná = název proměnné;
- trivializace = "'", (konstrukce | entita);
- kompozice = "[", volitelný bílý znak, konstrukce, volitelný bílý znak, konstrukce, { konstrukce }, volitelný bílý znak, "]" | konstrukce, "@wt";
- uzávěr = "[", volitelný bílý znak, lambda proměnné, volitelný bílý znak, konstrukce, volitelný bílý znak, "]";
- lambda proměnné = "\", volitelný bílý znak, typované proměnné;
- typované proměnné = název proměnné, volitelný bílý znak, [":", volitelný bílý znak, datový typ], { volitelný bílý znak,",", název proměnné, [volitelný bílý znak, ":", volitelný bílý znak, datový typ]};
- n-provedení = "^", volitelný bílý znak, nenulové číslo, volitelný bílý znak, (konstrukce | entita);
- entita = klíčové slovo | název entity | číslo | symbol;

- typ názvu = název s velkým písmenem na začátku;
- název entity = název s velkým písmenem na začátku;
- název proměnné = název s malým písmenem na začátku;
- název objektu = název s velkým písmenem na začátku;
- klíčové slovo = “ForAll” | “Exist” | “Every” | “Some” | “True” | “False” | “And” | “Or” | “Not” | “Implies” | “Equiv”;
- malé písmeno = “a” | “b” | … | “z”;
- velké písmeno = “A” | “B” | … | “Z”;
- symbol = “+” | “-“ | “*” | “/”;
- číslo = “0” | nenulové číslo;
- nenulové číslo = “1” | “2” | … | “9”;
- číslo = číslo, { číslo }[“.”, číslo, { číslo }];
- název s velkým písmenem na začátku = velké písmeno, { malé písmeno | velké písmeno | “_” | číslo };
- název s malým písmenem na začátku = malé písmeno, { malé písmeno | velké písmeno | “_” | číslo };
- bílý znak = bílý_znak, volitelný bílý znak;
- volitelný bílý znak = { bílý_znak };
- bílý_znak = mezera | tabulátor | nový řádek;

# 6 Případová studie

V této kapitole je popsána případová studie, jejímž cílem je vytvoření *agenta* implementující *symbolickou metodu strojového učení* využívající TIL jako formální jazyk pro reprezentaci *znalostí*.

Vize je následující:

*Cíle je vytvořit samostatného sociálního agenta, jehož funkcí bude tvorba hypotéz konceptů na základě symbolických metod strojového učení. Agent bude od svého spuštění naslouchat, dokud nějaký jiný agent nevyužije jeho služeb. Agentovi budou poskytovány pozitivní a negativní příklady konceptů a ontologie pro jejich zpracování. Na žádost ostatních agentů bude odesílat vytvořený model žadateli.*

> ***Definice 51:*** *Sociální agent je agent, který komunikuje s ostatními agenty pomocí vyššího komunikačního jazyka.*

Funkční požadavky na *agenta* jsou následující:

1. Agent bude přijímat pozitivní příklady.
2. Agent bude přijímat negativní příklady.
3. Bude mít uloženou aktuální hypotézy.
4. Agent se bude učit hypotézu pomocí symbolických metod strojového učení.
5. Agent bude umět na příklad vyprázdnit obsah hypotézy.
6. Agent na žádost odešle hypotézu žadateli.
7. Agent v případě prázdné hypotézy a žádosti o hypotézu, odešle žadateli chybovou zprávu.
8. Agent v případě prázdné hypotézy a přijatého negativního příkladu odešle žadateli chybovou zprávu.
9. Agent bude přijímat ontologie atributů a bude si je ukládat.
10. Agent bude komunikovat standardem FIPA.

Jednotlivé body funkčních požadavků jsou promítnuty do popisu funkcionality pomocí případů užití obsahující cíl případu užití, aktéry v něm vystupující, podmínky předcházející případ užití, podmínky následující případ užití, hlavní a alternativní scénář:

**1. Agent bude přijímat pozitivní příklady**

Cíl: Agent zpracuje pozitivní příklad konceptu zaslaným žadatelem. Pokud je aktuální hypotéza prázdná, vytvoří novou ze zaslaného příkladu, jestli bude model obsahovat rozpracovanou hypotézu, provede dle její generalizaci.

Aktéři:

- Agent-supervizor (žadatel)
- Agent-learner

Prekondice:

- Agent-learner je aktivní a připraven na příjem zpráv
- Agent-learner má prázdnou hypotézu nebo má hypotézu v procesu učení
- Agent-learner má uloženou ontologii atributů

Postkondice:

- Agent-learner má nově vytvořenou hypotézu nebo provedl generalizaci hypotézy na základě pozitivního příkladu a čeká na další zprávy

Hlavní scénář:

1. Agent-supervizor odešle agentovi-learnerovi pozitivní příklad
2. Agent-learner přijme pozitivní příklad
3. Agent-learner zkontroluje, zda má prázdnou hypotézu
4. Agent-learner má prázdnou hypotézu
5. Agent-learner ukládá přijatý pozitivní příklad jako novou hypotézu

Alternativní scénář:

1. Agent-supervizor odešle agentovi-learnerovi pozitivní příklad
2. Agent-learner přijme pozitivní příklad
3. Agent-learner zkontroluje, zda má prázdnou hypotézu
4. Agent-learner nemá prázdnou hypotézu
5. Agent-learner provádí generalizaci dle modelu učení

**2. Agent bude přijímat negativní příklady**

Cíl: Agent zpracuje negativní příklad konceptu zaslaným žadatelem. Pokud aktuální model nebude prázdný provede specializaci dle algoritmu učení, jinak neprovede nic.

Aktéři:

- Agent-supervizor (žadatel)
- Agent-learner

Prekondice:

- Agent-learner je aktivní a připraven na příjem zpráv
- Agent-learner má prázdnou hypotézu nebo má hypotézu v procesu učení
- Agent-learner má uloženou ontologii atributů

Postkondice:

- Agent-learner má specializovanou hypotézu a čeká na další zprávy

Hlavní scénář:

1. Agent-supervizor odešle agentovi-learnerovi negativní příklad
2. Agent-learner přijme negativní příklad
3. Agent-learner zkontroluje, zda má prázdnou hypotézu
4. Agent-learner nemá prázdnou hypotézu
5. Agent-learner provádí specializaci hypotézy dle modelu učení

Alternativní scénář:

1. Agent-supervizor odešle agentovi-learnerovi negativní příklad
2. Agent-learner přijme negativní příklad
3. Agent-learner zkontroluje, zda má prázdnou hypotézu
4. Agent-learner má prázdnou hypotézu
5. Agent-learner odesílá chybovou zprávu

**3. Agent bude umět vyprázdnit hypotézu**

Cíl: Agent po přijetí příkazu vymaže hypotézu v procesu učení.

Aktéři:

- Agent-supervizor (žadatel)
- Agent-learner

Prekondice:

- Agent-learner je aktivní a vnímá
- Agent-learner má hypotézu v procesu učení
- Agent-learner má uloženou ontologii atributů

Postkondice:

- Agent má prázdnou hypotézu a čeká na další zprávy

Hlavní scénář:

1. Agent-supervizor odešle agentovi-learnerovi příkaz pro vymazání hypotézy
2. Agent-learner přijme příkaz
3. Agent-learner zkontroluje, zda má prázdnou hypotézu
4. Agent-learner nemá prázdnou hypotézu
5. Agent-learner vymaže učenou hypotézu

Alternativní scénář:

1. Agent-supervizor odešle agentovi-learnerovi příkaz pro vymazání hypotézy
2. Agent-learner přijme příkaz
3. Agent-learner zkontroluje, zda má model prázdný
4. Agent-learner má prázdný model

**4. Agent na žádost odešle model**

Cíl: Agent na žádost odešle hypotézu žadateli.

Aktéři:

- Agent-supervizor (žadatel)
- Agent-learner

Prekondice:

- Agent-learner je aktivní a vnímá
- Agent-learner má hypotézu v procesu učení
- Agent-learner má uloženou ontologii atributů

Postkondice:

- Agent-learner agent má uloženou hypotézu a čeká na další zprávy

Hlavní scénář:

1. Agent-supervizor odešle agentovi-learnerovi negativní příklad
2. Agent-learner přijme negativní příklad
3. Agent-learner zkontroluje, zda má model prázdný
4. Agent-learner nemá prázdný model
5. Agent-learner odešle Agentu-supervizorovi hypotézu

Alternativní scénář:

1. Agent-supervizor odešle agentovi-learnerovi příkaz pro zaslání hypotézy
2. Agent-learner přijme příkaz
3. Agent-learner zkontroluje, zda má hypotézu prázdnou
4. Agent-learner má prázdnou hypotézu
5. Agent-learner agent odešle zprávu o prázdném modelu

**5. Agent bude přijímat ontologie atributů a bude si je ukládat**

Cíl: Rozšíření agentovi ontologie o nové informace

Aktéři:

- Agent-supervizor (žadatel)
- Agent-learner

Prekondice:

- Agent-learner je aktivní a vnímá
- Agent-learner má uloženou ontologii atributů

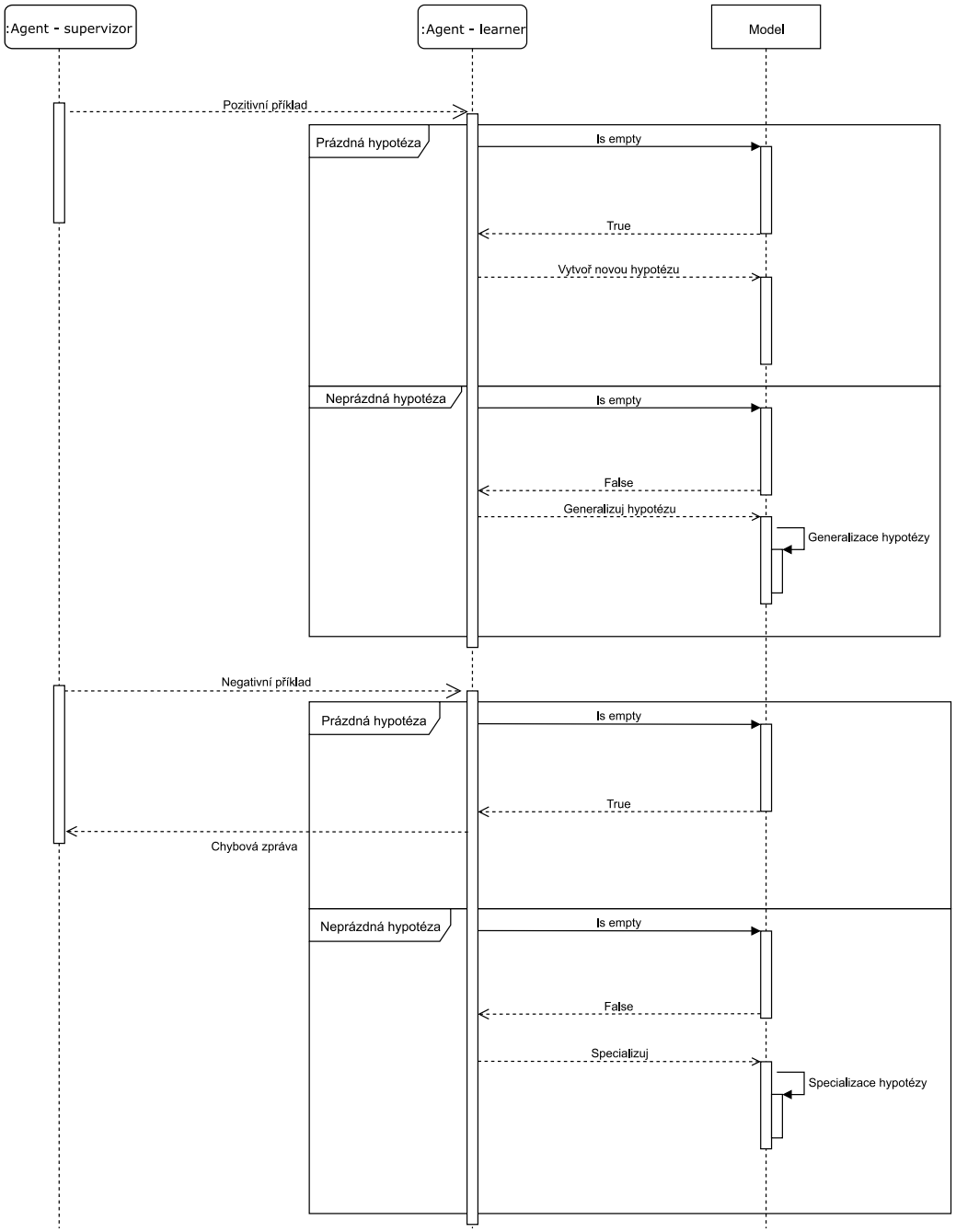
Postkondice:

- Agent-learner má obohacenou ontologii atributů

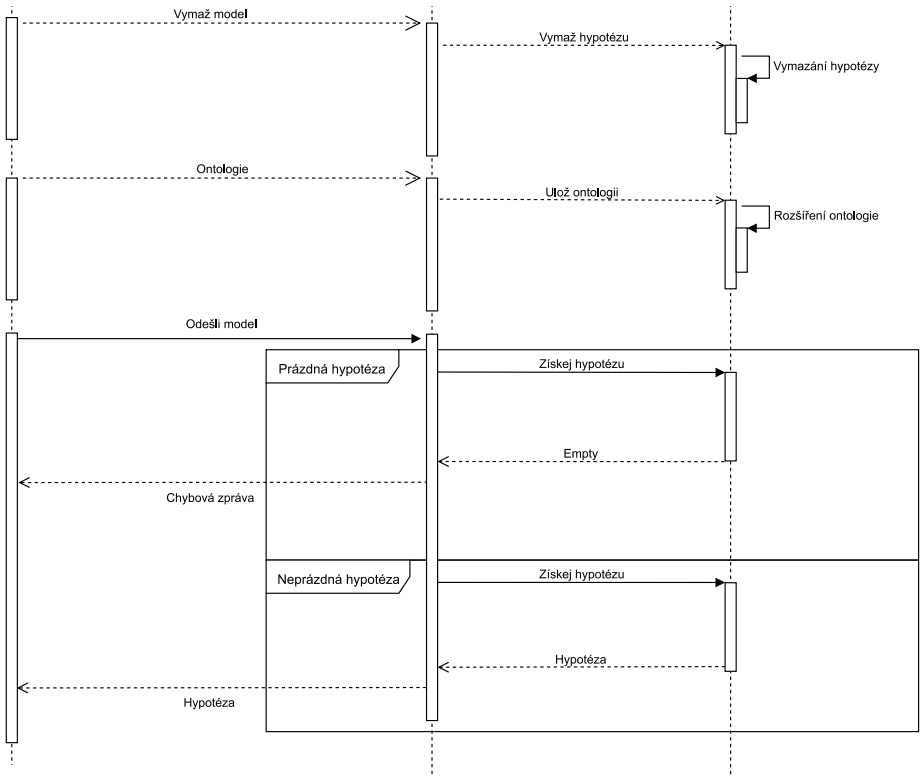
Hlavní scénář:

1. Agent-supervizor odešle agentovi-learnerovi ontologii
2. Agent-learner přijme ontologii
3. Agent-learner zkontroluje, zda má prázdnou ontologii
4. Agent-learner nemá prázdnou ontologii
5. Agent-learner agent si obohatí ontologii o nové informace

Komunikace mezi agentem-supervizorem a agentem-learnerem je znázorněna v čase pomocí sekvenčního diagramu na obrázku 6 a 7.

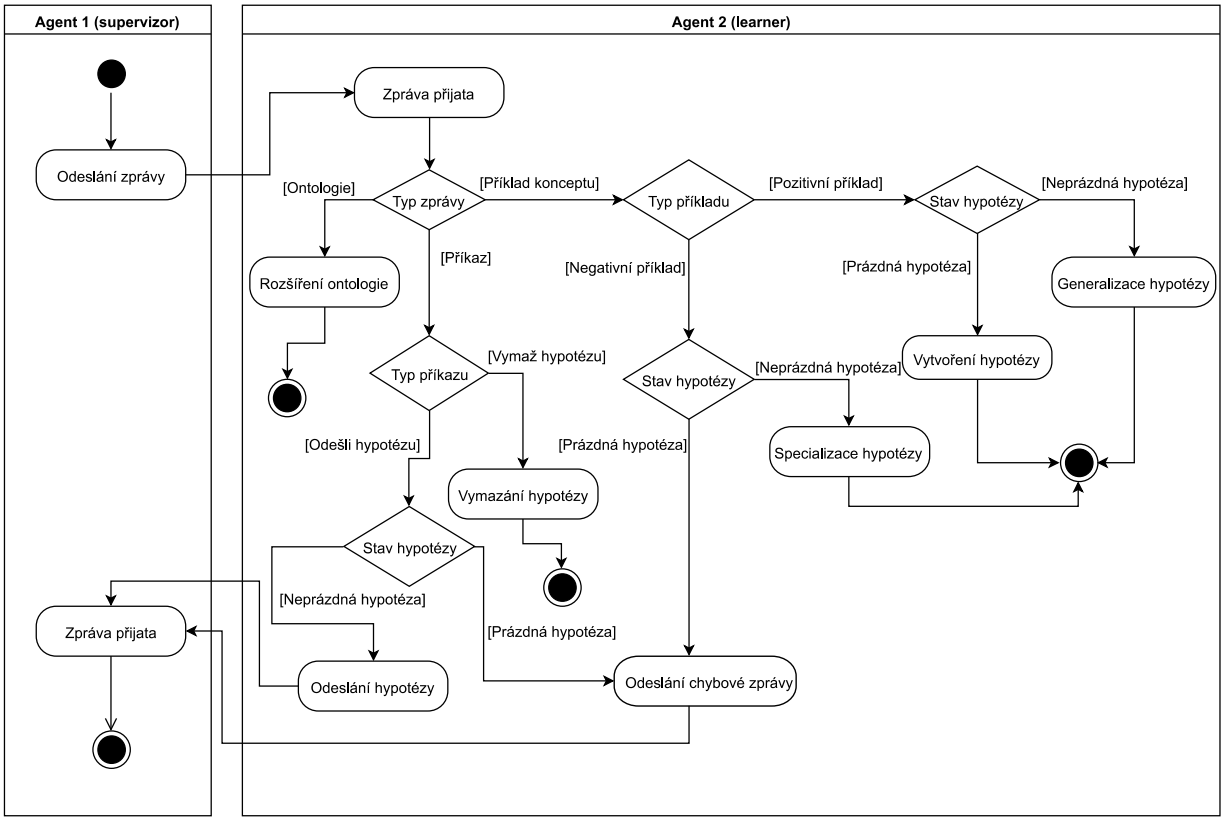



Obrázek 6: Sekvenční diagram komunikace agentů část 1.




Obrázek 7: Sekvenční diagram komunikace agentů část 2.

Algoritmus agenta je znázorněn pomocí diagramu aktivit na obrázku 8.




Obrázek 8: Diagram aktivit agenta

Obrázek 7 je detailněji zobrazen v příloze 1, obrázek 8 v příloze 2.

## 6.1 Model strojového učení

Jako model pro funkci učení agenta jsem vybral algoritmus vytvořený P. Winstonem publikovaný v [27].

V [4] se dočteme, že algoritmus, stejně jako všechny ostatní algoritmy symbolických metod strojového učení, lze popsat pomocí obecného frameworku skládajícího se z *cíle agenta*, *trénovacích dat*, *reprezentaci dat* a *sady operací manipulující s reprezentací*:

- **Cíl agenta:** Vytvořit *hypotézu konceptu*. Výsledná *hypotéza* je natolik obecná, aby při klasifikováním pomocí této *hypotézy* byl *agent* schopen identifikovat všechny pozitivní příklady *konceptu*, ale zároveň je natolik specifická, aby vyloučil všechny příklady, které danému *konceptu* neodpovídají.
- **Trénovací data:** *Agent* vytvoří *hypotézu* na základě symbolicky reprezentovaných pozitivních a negativních příkladů daného *konceptu*. Pozitivní a negativní příklady jsou poskytnuty jiným *agentem* představující supervizora.
- **Reprezentace dat:** Příklady jsou reprezentována pomocí TIL, která je zapsaná v jazyku *TIL-Script*.
- **Operace manipulující s reprezentací dat:** S reprezentací dat manipulují dvě hlavní operace: *generalizace* a *specializace*. Operace *generalizace* je spouštěna, pokud supervizor zašle *agentovi* pozitivní příklad, operace *specializace* je spuštěna, pokud supervizor zašle negativní příklad. *Generalizace* je obsahuje 4 heuristické funkce generalizující hypotézu, *specializace* obsahuje 2 heuristické funkce, které hypotézu specializují.

Operace *generalizace*, která generalizuje hypotézu na základě pozitivního příkladu, je v [27] specifikována následovně:

1. Porovnej model[16] a pozitivní příklad pro získání rozdílů.
2. Pro každý rozdíl proveď:
    a. Pokud model a pozitivní příklad obsahují shodný atribut z rozdílnými hodnotami, pak:
        i. Pokud hodnoty spadají do společné nejspecifičtější obecné třídy, použij heuristiku *climb-tree*.
        ii. Pokud hodnoty nemají společnou nejspecifičtější obecnou třídu, použij heuristiku *enlarge-set*.
        iii. Pokud se hodnoty navzájem vylučují, použij heuristiku *drop-link*.
    b. Pokud model obsahuje atribut, který se nenachází v pozitivním příkladu, použij heuristiku drop-link.
    c. Pokud model a pozitivní příklad obsahují shodný atribut z rozdílnými numerickými hodnotami, použij heuristiku *close-interval*.

[16] Model je hypotéza v procesu učení.

d. V případě jiných rozdílů ignoruj příklad.

Operace *specializace*, která hypotézu specializuje pomocí negativního příkladu konceptu, je v [27] specifikována následovně:

1. Porovnej model a negativní příklad pro získání rozdílu.
2. Pokud mezi modelem a negativním příkladem existuje pouze jeden rozdíl, pak:
    a. Pokud model obsahuje atribut, který se nenachází v negativním příkladě, použij heuristiku *require-link*.
    b. Pokud model neobsahuje atribut, který se nachází v negativním příkladě, použij heuristiku *forbid-link*.
    c. V jiném případě příklad ignoruj.

Heuristické funkce jsou specifikovány následovně:

Typy požité v příkladech:

$Colour, Pneumatika, Stav\_motoru/((o\iota)_{\tau\omega}\iota)_{\tau\omega}$

$Počet\_dveří, Počet\_kol/\left((o\tau)_{\tau\omega}\iota\right)_{\tau\omega}$

$Kobaltová, Indigo, Modrá, Zimní, Letní, Nefunkční\ /(o\iota)_{\tau\omega}$

$True/(oo_{\tau\omega})_{\tau\omega}$

$=/(o(o\iota)_{\tau\omega}\ (o\iota)_{\tau\omega})$

$\geq, \leq/(o(o\tau)_{\tau\omega}\ \tau)$

- **Climb-tree:** Pokud má hodnota atributu v modelu a pozitivním příkladu společnou nejspecifičtější obecnou třídu[17], hodnota atributu v modelu je nahrazena touto třídou, například

  Model obsahuje individuum x s atributem *barva* s hodnotou *kobaltová*:

  $$\ldots \wedge \left[ {}^0= \left[ {}^0Colour_{wt}\ x\right]\ {}^0Kobaltová\right] \wedge \ldots$$

  Pozitivní příklad obsahuje individuum x s atributem *barva* s hodnotou *indigo*:

  $$\ldots \wedge \left[ {}^0= \left[ {}^0Colour_{wt}\ x\right]\ {}^0Indigo\right] \wedge \ldots$$

  Hodnota atributu v modelu je nahrazena společnou nejspecifičtější třídou *modrá*, atribut je označen jako MUST-BE[18]:

  $$\ldots \wedge \left[ {}^0True_{\ wt}\ \lambda w\lambda t\left[ {}^0= \left[ {}^0Colour_{wt}\ x\right]\ {}^0Modrá\right]\right] \wedge \ldots$$

[17] Informace je obsažena v ontologii poskytnuté supervizorem

[18] MUST-BE označení slouží pro označení atributů, které jsou pro daný koncept klíčové. Při použití hypotézy pro klasifikování neviděných příkladů lze porovnat, zda příklad obsahuje všechny MUST-BE označené atributy v hypotéze. Pokud obsahuje, příklad je instancí daného konceptu. V TIL je MUST-BE označení realizováno pomocí vlastnosti propozic *True* popsané v [21].

- **Enlarge-set:** Pokud nemá hodnota atributu v modelu a pozitivním příkladu společnou nejspecifičtější obecnou třídu, hodnota atributu v modelu je nahrazena unionem dvou hodnot, například;

  Model obsahuje individuum x s atributem *pneumatika* s hodnotou *zimní*:

  $$\ldots \wedge \left[ {}^{0}= \left[ {}^{0}Pneumatika_{wt}\ x \right]\ {}^{0}Zimní \right] \wedge \ldots$$

  Pozitivní příklad obsahuje individuum x s atributem *pneumatika* s hodnotou *letní*:

  $$\ldots \wedge \left[ {}^{0}= \left[ {}^{0}Pneumatika_{wt}\ x \right]\ {}^{0}Letní \right] \wedge \ldots$$

  Hodnota atributu v modelu je nahrazena unionem hodnot *zimní* a *letní*, atribut je označen jako MUST-BE:

  $$\ldots \wedge \left[ {}^{0}True_{\ wt}\ \lambda w \lambda t \left[ \left[ {}^{0}= \left[ {}^{0}Pneumatika_{wt}\ x \right]\ {}^{0}Letní \right] \right.\right.$$
  $$\left.\left. \vee \left[ {}^{0}= \left[ {}^{0}Pneumatika_{wt}\ x \right]\ {}^{0}Zimní \right] \right] \right] \wedge \ldots$$

- **Drop-link:** Pokud model obsahuje atribut, který se nevyskytuje v pozitivním příkladu (atribut je nadbytečný) nebo se hodnoty společného atributu navzájem vylučují (světla svítí/nesvítí), atribut se z modelu odstraní.

- **Close-interval:** Pokud model a pozitivní příklad obsahují společný atribut s rozdílnými numerickými hodnotami, hodnota v modelu je nahrazena intervalem pokrývající obě rozdílné hodnoty, například

  Model obsahuje individuum x s atributem *počet_dveří* s hodnotou *3*:

  $$\ldots \wedge \left[ {}^{0}=_{\tau}\ \left[ {}^{0}Počet\_dveří_{wt}\ x \right]\ {}^{0}3 \right] \wedge \ldots$$

  Pozitivní příklad obsahuje individuum x s atributem *počet_dveří* s hodnotou *5*:

  $$\ldots \wedge \left[ {}^{0}=_{\tau}\ \left[ {}^{0}Počet\_dveří_{wt}\ x \right]\ {}^{0}5 \right] \wedge \ldots$$

  Hodnota atributu v modelu je nahrazena intervalem s hodnotou *<3;5>*, atribut je označen jako MUST-BE:

  $$\ldots \wedge \left[ {}^{0}True_{\ wt}\ \lambda w \lambda t \left[ \left[ {}^{0}\geq \left[ {}^{0}Počet\_dveří_{wt}\ x \right]\ {}^{0}3 \right] \wedge \left[ {}^{0}\leq \left[ {}^{0}Počet\_dveří_{wt}\ x \right]\ {}^{0}5 \right] \right] \right]$$
  $$\wedge \ldots$$

- **Require-link:** Pokud model obsahuje atribut, který se nenachází v negativním příkladu, atribut je převeden na MUST-BE formu, například:

  Model obsahuje individuum x s atributem *počet_kol* s hodnotou 4:

  $$\ldots \wedge \left[ {}^{0}=_{\tau}\ \left[ {}^{0}Počet\_kol_{wt}\ x \right]\ {}^{0}4 \right] \wedge \ldots$$

  Negativní příklad tento atribut neobsahuje, atribut v modelu je označen jako MUST-BE:

  $$\ldots \wedge \left[ {}^{0}True_{\ wt}\ \lambda w \lambda t \left[ {}^{0}=_{\tau}\ \left[ {}^{0}Počet\_kol_{wt}\ x \right]\ {}^{0}4 \right] \right] \wedge \ldots$$

- **Forbid-link:** Pokud model neobsahuje atribut, který se vyskytuje v negativním příkladu(atribut nechtěný v konceptu), atribut je přidán do modelu a označen jako MUST-NOT-BE (v TIL realizováno pomocí negace):

  Negativní příklad obsahuje atribut *stav_motoru* s hodnotou *nefunkční:*

  $$\dots \wedge \left[ {}^0= \left[ {}^0Stav\_motoru_{wt}\ x \right]\ {}^0Nefunkční \right] \wedge \dots$$

  Atribut je přidán do modelu v MUST-NOT-BE formě:

  $$\dots \wedge \left[ {}^0\neg\left[ {}^0= \left[ {}^0Stav\_motoru_{wt}\ x \right]\ {}^0Nefunkční \right] \right] \wedge \dots$$

Příklad učení je popsán v [22] na příkladu konceptu oblouku. Detailnější popis modelu naleznete v [27].

## 6.2 Komunikace mezi agenty

Komunikace s ostatními agenty je realizována pomocí standardu pro multiagentní komunikaci ACL vyvinutý společností FIPA dostupný v [28].

Zprávy, kterou si agenti zasílají, jsou složeny ze čtyř částí:

- **Typ komunikativní komunikativního aktu:** Určuje typ komunikativního aktu. Má jediný parametr *performative*, nabývající v našem přídě hodnot *request* (odesílatel žádá po příjemci vykonání nějaké akce), *failure* (akce zpravující agenta o neúspěchu provedené akce), *confirm* (odesílatel informuje příjemce, že daná propozice je pravdivá)
- **Účastníci komunikace:** Identifikuje účastníky komunikace. Má dva parametry, *sender* a *reciever*. Hodnoty parametrů jsou identifikátory agentů, v našem případě IP adresy agentů.
- **Obsah zprávy:** Označuje samotný obsah zprávy. Má jediný parametr *content*. Obsah zprávy je ontologie, pozitivní příklad učeného konceptu, negativní příklad učeného, konceptu hypotéza naučeného konceptu, příkaz pro zaslání nebo vymazání hypotézy nebo chybová zpráva.
- **Popis obsahu:** Specifikuje obsah zprávy. V našem případě má jediný parametr *ontology*, který specifikuje, jaký obsah je zprávou odeslán. Hodnoty mohou být *ontology* (pokud obsahem zprávy je ontologie), *positive_ex* (označující pozitivní příklad učeného konceptu), *negative_ex* (označující negativní příklad), *model* (označující hypotézu konceptu), *command* (označující příkaz), *error* (označující chybovou zprávu).

Příklady zpráv v komunikaci mezi dvěma agenty, kde agent `192.0.0.1` posílá agentovi `192.0.0.2` ontologii pro funkci heuristiky *climb-tree*, by vypadala takto:

```
(request
  :sender 192.0.0.1
  :receiver 192.0.0.2
  :content
    "['Blue@wt 'Cobalt].

     ['Blue@wt 'Indigo].

     ['Colour@wt 'Blue].

    "
  :ontology ontology)
```

## 6.3 Implementace agenta

Zdrojové kódy agenta se nacházejí v příloze 3. Návod na použití programu se nachází v příloze 4.

# Závěr

V této práci jsem shrnul strojové učení ze dvou pohledů spojených se *symbolickými metodami strojového učení*. Zpětná vazba a model funkce učení jsou parametry, které nejlépe rozdělují jednotlivé *metody symbolického učení* podle svého účelu. Při rozhodování, kterou symbolickou metodu v programu použít, je vhodnější nejprve určit, s jakým typem zpětné vazby bude program pracovat. A dále vybrat model funkce učení.

Pokud nás zajímá, jakou informaci ze sady dat dokáže program získat, pak modely *učení bez učitele* jsou tím vhodným nástrojem. Pomocí *konceptuálního shlukování* objektů nebo individuí získáme z neklasifikovaných dat obecné popisy prvků, které se v daném shluku nacházejí.

Pokud bychom chtěli rozšířit znalosti programu o popis objektů nebo individuí, pak *konceptuální učení s učitelem* nám nabízí k tomuto účelu několik variant. Pokud pracujeme s přesně definovanými daty a zajímá nás, jaké všechny *konzistentní hypotézy* popisují zpracovaná data, použijeme jeden z modelů hledání hypotéz pomocí *Version space*. Pokud pracujeme s daty, které nejsou plně definovány a mohou obsahovat šum, můžeme vytvořit hypotézu pomocí modelu *rozhodovacího stromu*, který sice uchovává pouze jednu *hypotézu konzistentní* na datech, dokáže však zpracovat data obsahující šum.

Symbolické metody pracují se zápisem zkušeností a *znalostí* v podobě *formálního jazyka*. Jednou z možností pro takový zápis je Transparentní intenzionální logika schopná sémanticky reprezentovat obsah přirozeného jazyka. V této práci shrnuta základní teorie *procedurální sémantiky*, která přiřazuje výrazům funkci konstruující jejich denotát, dále práce obsahuje *hierarchické rozdělení entit* tvořící denotát výrazu a další prvky TIL související s reprezentací znalostí. Pro zpracování TIL programem je uvedena syntax jazyka *TIL-Script*, který je operačně-izomorfní s TIL. Syntax obsahuje *bázi typů*, zápis *konstrukcí* a *gramatiku* jazyka *TIL-Script*.

V případové studii je popsána analýza a návrh *agenta* schopného budovat *hypotézu konceptu*, zapsanou v TIL, pomocí symbolické metody strojového učení. Hypotéza je induktivně budována z pozitivních a negativních příkladů *konceptu*, zaslaných ve zprávách *standardu FIPA*, jinými agenty. Agent na žádost odesílá naučenou hypotézu, a tím rozšiřuje bázi znalostí agentů v systému a zvýší inteligenci celého systému.

Námětem další práce je rozšíření schopností o *agenta* o ukládání negativních příkladů v paměti *agenta* v případě, kdy má *agent* prázdnou hypotézu. Negativní příklady by vhodně specializovaly *hypotézu* v okamžiku, kdy by se jako nové *hypotéza* uložil první pozitivní příklad. Tato funkcionalita by musela zhodnotit relevanci negativních příkladů vzhledem k učenému *konceptu*, protože by bylo nežádoucí uchovávat v paměti nerelevantní negativní příklady. Další rozšiřující funkcionalitou by mohla být práce s ontologií *agenta*. *Agent* by ontologii sdílet s ostatními *agenty*, stejně jako *hypotézy*. Dalším příkladem rozšíření práce by mohl být klasifikační algoritmus, který by sloužil pro klasifikaci příkladů zaslanými jinými agenty.

# Reference


[1] MITCHELL, Tom. *Machine Learning*. New York: McGraw-Hill, 1997. ISBN 00-704-2807-7.

[2] POOLE, David a Alan MACKWORTH. *Artificial intelligence: foundations of computational agents*. 2nd pub. Cambridge: Cambridge University Press, 2010. ISBN 978-0-521-51900-7.

[3] RUSSELL, Stuart a Peter NORVIG. *Artificial intelligence: a modern approach*. 2nd ed. Harlow: Pearson Education, 2014. ISBN 978-1-29202-420-2.

[4] LUGER, George. *Artificial intelligence: structures and strategies for complex problem solving*. 6th ed. Boston: Pearson Addison-Wesley, 2009. ISBN 978-0-321-54589-3.

[5] TAN, Pang-Ning, Michael STEINBACH, Anuj KARPATNE a Vipin KUMAR. *Introduction to data mining*. Second edition. NY NY: Pearson, 2019. ISBN 978-013-3128-901.

[6] NEAL, Radford. *Statistical Methods for Machine Learning and Data Mining* [online]. In: . b.r. [cit. 2018-05-29]. Dostupné z: http://www.utstat.utoronto.ca/~radford/sta414.S11/week1a.pdf

[7] SUTTON, Richard a Andrew BARTO. *Reinforcement learning: an introduction*. Second edition. Cambridge, Massachusetts: The MIT Press, 2018. ISBN 978-026-2039-246.

[8] KRIESEL, David. *A Brief Introduction to Neural Networks* [online]. 2007 [cit. 2019-03-29]. Dostupné z: http://www.dkriesel.com

[9] NIELSEN, Michael. *Neural Networks and Deep Learning* [online]. 2015. Determination Press, 2015 [cit. 2019-03-29]. Dostupné z: http://neuralnetworksanddeeplearning.com

[10] BHASIN, Harsh a Surbhi BHATIA. Application of Genetic Algorithms in Machine learning. *International Journal of Computer Science and Information Technologies* [online]. 2011, (2), 4 [cit. 2019-04-16]. ISSN 0975-9646. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.227.9827

[11] CARR, Jenna. An Introduction to Genetic Algorithms. In: *Semantic Scholar* [online]. b.r. [cit. 2019-03-30]. Dostupné z: https://www.semanticscholar.org/paper/An-Introduction-to-Genetic-Algorithms-Carr/e9f8d49686a4c8d99d0a5ceba85c4508c30d57c4

[12] NEAPOLITAN, Richard. *Learning Bayesian Networks*. 1. vydání. Prentice Hall, 2003. ISBN 9780130125347.

[13] KUČERA, Antonín. *Matematická logika: Materiály ke kurzu MA007* [online]. In: . b.r. [cit. 2019-04-03]. Dostupné z: https://www.fi.muni.cz/usr/kucera/teaching.html

[14] SUN, R. *Artificial Intelligence: Connectionist and Symbolic Approaches* [online]. Elsevier, 2001, , 783-789 [cit. 2019-03-31]. DOI: 10.1016/B0-08-043076-7/00553-2. ISBN 9780080430768. Dostupné z: https://linkinghub.elsevier.com/retrieve/pii/B0080430767005532

[15] KUBÍK, Aleš. *Inteligentní agenty*. Vyd. 1. Brno: Computer Press, 2004. ISBN 80-251-0323-4.

[16] WEISS, Gerhard. *Multiagent systems*. Second edition. Cambridge, Massachusetts: The MIT Press, 2013. ISBN 978-026-2018-890.

[17] FRANK, Eibe. *Pruning Decision Trees and Lists*. Department of Computer Science, University of Waikato, 2000. Disertační práce.

[18] FISHER, D.H. Knowledge acquisition via incremental conceptual clustering. *Machine learning*. 1987, (2), 139–172. DOI: https://doi.org/10.1007/BF00114265. ISSN 0885-6125.

[19] MICHALSKI, R.S, J.G CARBONELL a T.M. MITCHELL. *Machine Learning: An Artificial Intelligence Approach*. 1. Berlin: Springer, 1983. ISBN 978-3-662-12405-5.

[20] TICHÝ, Pavel. *The foundations of Frege's logic*. 1. vydání. New York: de Gruyter, 1988. ISBN 31-101-1668-5.

[21] DUŽÍ, Marie a Pavel MATERNA. *TIL jako procedurální logika: Průvodce zvídavého čtenáře Transparentní intensionální logikou*. 1. vydání. Bratislava: Aleph, 2012. ISBN 978-80-89491-08-7.

[22] MENŠÍK, Marek, Marie DUŽÍ, Adam ALBERT, Vojtěch PATSCHKA a Miroslav PAJR. *Machine learning using TIL*. In: . To appear in Proceedings of the 29th International Conference on Information Modelling and Knowledge Bases, EJC 2019, b.r.

[23] DUŽÍ, Marie, Bjørn JESPERSEN a Pavel MATERNA. *Procedural semantics for hyperintensional logic: foundations and applications of transparent intensional logic*. New York: Springer, 2010. ISBN 978-90-481-8812-3.

[24] DUŽÍ, Marie. *Zpracování přirozeného jazyka (TIL)* [online]. b.r. [cit. 2019-04-16]. Dostupné z: http://www.cs.vsb.cz/duzi/TIL.html

[25] DUŽÍ, Marie, Marek MENŠÍK, Miroslav PAJR a Vojtěch PATSCHKA. Natural Deduction System in the TIL-Script Language. *Information Modelling and Knowledge Bases XXX*. 2019, **312**, 237-255. DOI: 10.3233/978-1-61499-933-1-237.

[26] CIPRICH, Nikola, Marie DUŽÍ a Michal KOŠINÁR. The TIL-Script Language. In: *Proceedings of the 2009 conference on Information Modelling and Knowledge Bases XX*. Amsterdam: IOS Press, 2009, s. 166-179. ISBN 978-1-58603-957-8.

[27] WINSTON, Patrick Henry. *Artificial intelligence*. 3rd ed. Reading, Mass.: Addison-Wesley Pub. Co., 1992. ISBN 02-015-3377-4.

[28] FIPA ACL Message Structure Specification. *Foundation for intelligent physical agents* [online]. FIPA TC Communication, b.r. [cit. 2019-04-21]. Dostupné z: http://www.fipa.org/specs/fipa00061/SC00061G.html

# Seznam příloh